Handbook

実用

UNIX ハンドブック

【改訂新版】

舟本 奨＝著

ナツメ社

cover design——nobuo nakagaki+yoichi matsuda

実用 UNIX ハンドブック

【改訂新版】

舟本 奨＝著

ナツメ社

はじめに

誕生してから25年以上の歴史を積み重ねているオペレーティングシステム(OS)のUNIXは、多種多様なコンピュータハードウェアで利用できるという面で、現在では他に類のない重要な位置を占めるようになってきています。この期間には、実に様々な機能が強化され、その時代のニーズに応えるべくさらに発展してきています。

ところが、この機能の豊富さがUNIXをこれからはじめようとする初心者にとっては、敷居を高く感じさせる要因となっています。例えば、コマンドについて言えば、コンピュータメーカー各社から提供されるUNIX製品の場合、その数は優に300を超え、それぞれのコマンドに多くのオプションがあります。これでは、的確なコマンドを選ぶだけでも相当な専門知識が必要です。そこで本書では、実用性をそこなわない範囲でコマンドや機能を精選し、わかりやすい解説をするように心掛けました。

本書は、多くのユーザ諸氏にご支持いただいた「実用UNIXハンドブック」の改訂第2版です。今回さらに実用性を高めるために、基本操作解説の増補、コマンドリファレンスへのコマンドの追加、対応するOSの追加、エディタリファレンスへのEmacsの追加、さらに、付録には第2レベルコマンドリファレンス、UNIX用語解説、MINIXコマンドとの対応表を新たに追加しました。

なお、執筆にあたっては、UNIX製品の提供メーカー各社（日本サンマイクロシステムズ、富士通、YHP、ソニー、IBM、DEC、日本電気、オムロン、ノベル）から、各種資料の提供を快諾していただきました。コマンドの使用頻度調査、掲載コマンドの選定、関連資料の整理などの作業で、上野有世さんをはじめとしたメンバーの方々のお世話になりました。第1版に対しては、多くの読者の方から有益な数多くのご指摘をいただきました。また、ナツメ出版企画の甲斐健一、山路和彦の両氏には、本書出版の機会を与えていただくと同時に、辛抱強く改版原稿を待っていただきました。最後になりますが、ここにこれら多くの個人・法人の方々に深く感謝いたします。

平成6年1月　　　　舟本　奨

■本書の構成

第1章　UNIXの基礎知識と基本操作

これからUNIXを使う方にも、基本的な機能（コマンド）の概念とそのオペレーションを一通り解説しています。この内容をマスターすれば、あとは第2章以降を参照すれば、より高度な使用法をマスターできます。

第2章　コマンド・リファレンス

入門から中級レベルのユーザに必要とされるコマンドの基本的な機能を説明したリファレンスマニュアルです。具体的な入力例とその解説を参照しながらコマンドの機能を理解できるようにしました。

第3章　Cシェル・リファレンス

UNIXの重要なサブシステムCシェルの機能（コマンド）リファレンスです。利用頻度の高いコマンドは、第2章に一般のコマンドとともに解説していますが、ここにはそれ以外のコマンドを中心に掲載しました。

第4章　vi/emacsエディタ・コマンド・リファレンス

テキストエディタのviとEmacsのサブコマンドを解説しました。viとEmacsの機能を対応させて参照できます。

付　録

UNIXコマンドの機能一覧表、第2章掲載コマンドの次に重要なコマンドのリファレンス、UNIX用語の解説、MINIXコマンドとの対照表、CTRLキーの入力の種類を掲載しました。

●コマンドに対する使用可能なUNIX一覧

シンボル	UNIX名	メーカー
SVR	System V系	
BSD	BSD系	
SUN	SunOS	サンマイクロシステムズ、富士通、東芝
HPU	HP-UX	ヒューレッドパッカード (HP)
NWS	NEWS-OS	ソニー
AIX	AIX	IBM
DEC	ULTRIX	DEC
EWS	EWS-UX	日本電気
OMR	UniOS	オムロン
SOL	Solaris	サンマイクロシステムズ、富士通、東芝
PCU	PC-UX	日本電気
UXW	UnixWare	ノベル

■コマンド・リファレンスの見方

イタリック表記の文字列には、適切な文字列を入れる。

* UNIX オペレーティングシステムは、米国 AT&T のベル研究所が開発し、AT&T がライセンスしています。
* 4.2BSD および 4.3BSD は、カリフォルニア州立大学バークレイ校で開発されたオペレーティングシテスムです。
* X Window System は、マサチューセッツ工科大学の商標です。
* NFS は、Sun Microsystems, Inc.の商標です。
* Ethernet は、XEROX 社の登録商標です。
* その他、本書に登場するシテスム名称、製品の名称等は、一般に開発メーカーの登録商標です。

目　次

第2章　コマンド・リファレンス

第3章　Cシェル・リファレンス

第4章　vi/emacsエディタ・コマンド・リファレンス

付　録

第1章

UNIXの基礎知識と基本操作

1・1

UNIXとは

UNIX(ユニックス)は、現在最もポピュラーなオペレーティングシステム(OS)です。パソコンではMS-DOSやWindows、メインフレームではIBM系OS(MVS、MSPなど)のように、実際にはこのUNIXよりも数多くのユーザを獲得しているものもあります。ところがこれらのOSの場合は、使えるコンピュータを限定してしまうのです。例えばMVSのような大規模なOSをパソコン上で使うことはほとんど不可能ですし、MS-DOSをメインフレーム上で動作させるということも意味のあることではありません。

ところがUNIXは、ワークステーションやミニコンをはじめとして、パソコンからメインフレーム、そしてスーパーコンピュータまでのあらゆる種類のコンピュータで使えます。たとえメーカーが異なっても大方は同様の操作ができるという非常に移植性の高いOSなのです。UNIXが

オープンシステム

といわれる所以がここにあるのです。

このUNIXには、次にあげるような特徴があります。

- **会話型マルチユーザ/マルチタスク**
- **階層構造ファイルシステム**
- **柔軟性の高いプロセスシステム**
- **柔軟性の高いコマンドインタプリンタ(シェル)**
- **豊富な開発コマンドやソフトウェアツール**
- **ウィンドウ環境**
- **ネットワーク機能**
- **ポータビリティ**

これらが数多くのユーザから支持を集める理由になっているのです。

会話型マルチユーザ／マルチタスク

UNIXを機能面から端的に表現すれば、

会話型

の

マルチユーザ／マルチタスク環境

を提供するOSということになります。

OSは、ハードウェアとユーザの仲介をするための基本ソフトウェアです。このハードウェアを有効に利用するためには、1台のハードウェアに対して複数のユーザが同時に使えるようにするためのマルチユーザ環境が必要になります。実際には、

TSS（Time Sharing System）

といって、コンピュータの処理装置すなわちCPUの占有時間を細かく区切る

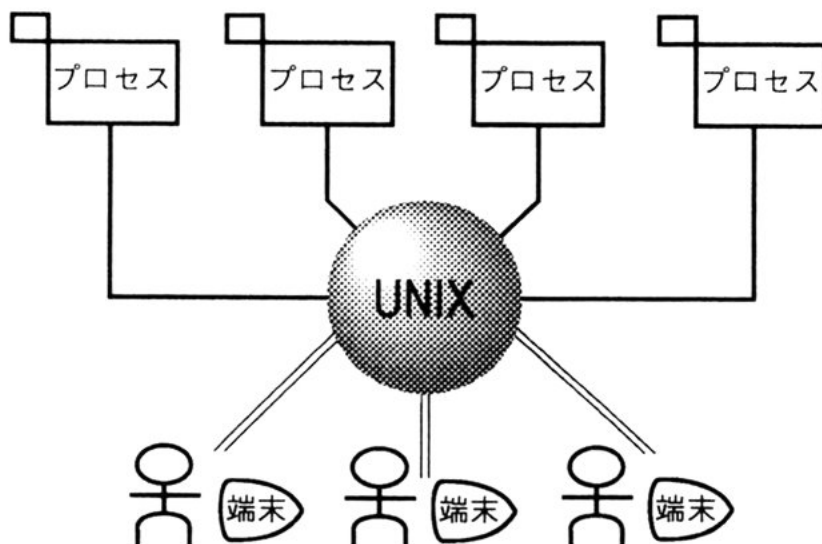
時分割方式

を使って、各ユーザには個別のハードを占有しているかのように感じさせる環境を実現しています。

このマルチユーザ環境では、CPUだけでなくディスクをはじめとした各種の資源を複数のユーザが共有することになります。これをうまく利用すればユーザの間での通信や情報の交換を行いながら、共同作業をすることもできるのです。

また、各ユーザが同時に複数の作業を実行することができる環境にもなっています。これはUNIXが

マルチタスク処理

を行っているからです。シングルタスクのMS-DOSのように1つのタスクがすべての資源を占有するわけではありません。時間のかかるプリンタへの出力処理を行っている間に、ワープロを使って別の文書処理を行ったり、プログラムのコンパイルを並行して実行させることができます。

階層構造によるファイル管理

UNIXでは、ユーザのデータを保持するファイルを階層構造状に管理しています。これは、MS-DOSに採用されている方式とほぼ同じです。同じ目的をもったファイルをまとめるために

ディレクトリ

というファイルの名前をグループ化して保持している特別なファイルを使っています。このディレクトリを使えば、用途や所有者ごとにファイルを分類して管理することができますから、機能的なファイル管理をすることができます。

UNIXでは、システム(OS)のファイルであるかユーザのファイルであるかにはかかわらず、すべてのファイルが1つの階層構造をなす

ファイルシステム

の中に配置されています。同じシステムを利用しているすべてのユーザの資産が共有されることになりますが、ユーザごとに利用範囲の制限がされています

ので、機密の保護についても問題はありません。

コマンドインタプリンタ(シェル)

ユーザから入力されたコマンドは、ユーザとUNIXの核の部分(ここでコマンドを実行する)の仲介を行う役割をはたすサブシステム、すなわち

シェル (Shell)

があってはじめて実行されます。つまりシェルは、入力されたコマンドを解釈して、これをUNIXの中核部分の

カーネル (Kernel)

に実行依頼します。シェルにコマンドの実行を依頼するときの形式は、人間が記述しやすいように工夫されています。これを、シェルはカーネルが処理できるいわばコンピュータに機能が理解できる形式に変換して引き渡します。

UNIXのシェルには、おのおのユーザが自分の使いやすいコマンドをつくったり、環境を自分用に整備するために必要な

プログラミング

機能もあります。1つのUNIXシステムには、カーネルは1つしかありませんが、シェルはユーザごとに最低1つずつ存在しますから、自分の好みに合わせて変更することができるわけです。

代表的なシェルには、BシェルとCシェルと呼ばれるものがあります。

豊富な開発コマンドやツール

UNIXでは、プログラム開発を行う際に基本的に必要とされるエディタ、コンパイラ、デバッガなどが通常標準で添付されています。つまり、いちいち買いそろえる必要はありません。UNIXは、もともと開発志向性の強いOSだったせいもあって、開発にかかわる各種のツールには不自由しません。

●代表的なプログラム開発コマンド

vi	visual editor	フルスクリーンエディタ
cc	c compiler	C 言語コンパイラ
ld	linkage editor	リンケージエディタ
cb	C Program beautifier	ソースプログラムの整形
indent	indentify files	ソースプログラムの字下げ・整形
lint	lint	プログラムの文法チェック
make	make object from source	コンパイル手順の自動化
dbx	extended debugger	ソースレベルデバッガ

ウィンドウ環境

UNIXでは、マルチウィンドウを実現するためのソフトウェアとしてMIT(マサチューセッツ工科大学) が提供している

X Window System (通常単にXと呼ばれることが多い)

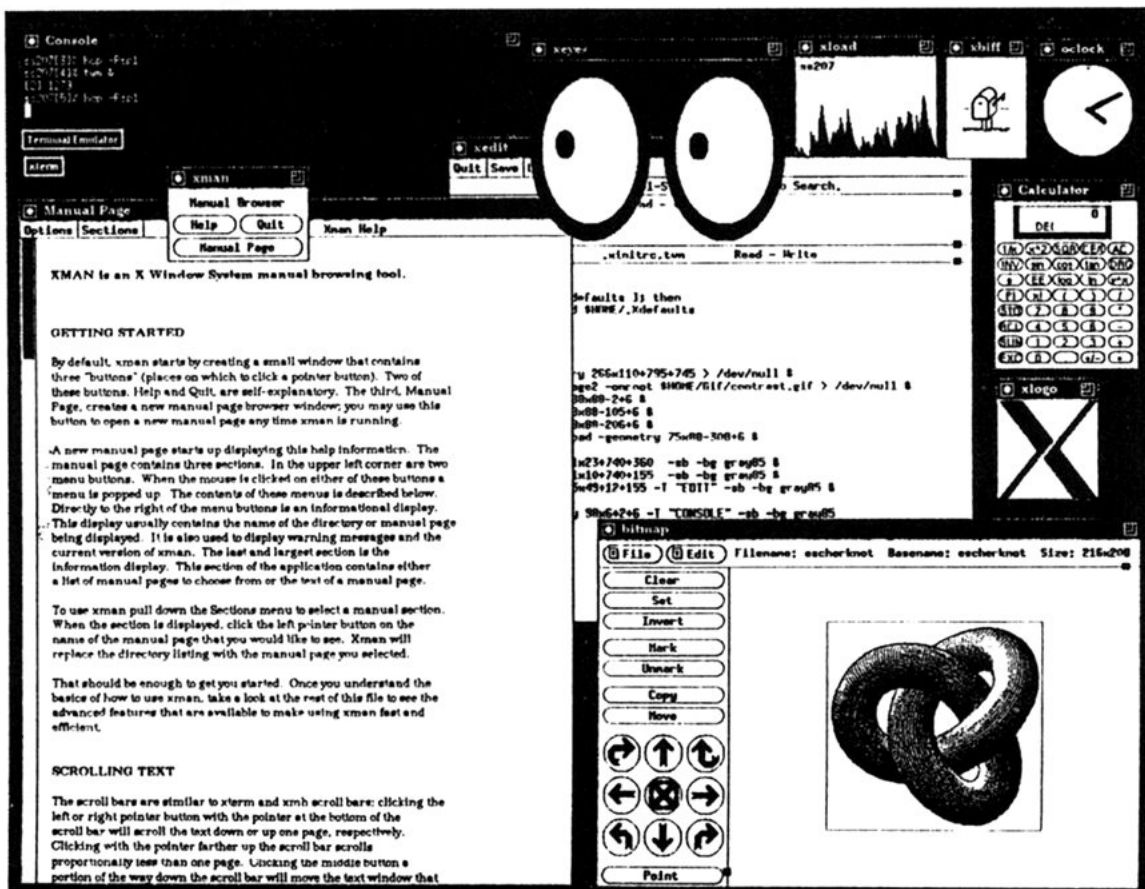
が標準的に使われています。

マルチウィンドウ環境では、同時に複数のプログラムの表示を行うことができますし、図形や画像といったグラフィック情報も表示できるようになっています。こうした環境は

GUI (Graphical User Interface)

と呼ばれています。

Xがメーカーや機種を問わず標準的に使われるいわゆる業界標準になってきたために、ユーザにとっては異なったメーカーのUNIXやアプリケーションを使うときにも、基本的にはXというウィンドウ環境を利用しますから、共通的な操作で済んでしまうという、つごうの良い環境になってきています。

●X Window System の画面

ローカルエリアネットワーク(LAN)

ワークステーションは１人のユーザが１台を占有することを基本としていますから、相互のユーザは

LAN（Local Area Network）

というネットワークを経由して、コミュニケーションを行います。LANを使えば、物理的にはおのおの独立したワークステーション上の個別の作業環境を統合化することができます。具体的には相互にデータを送受信する

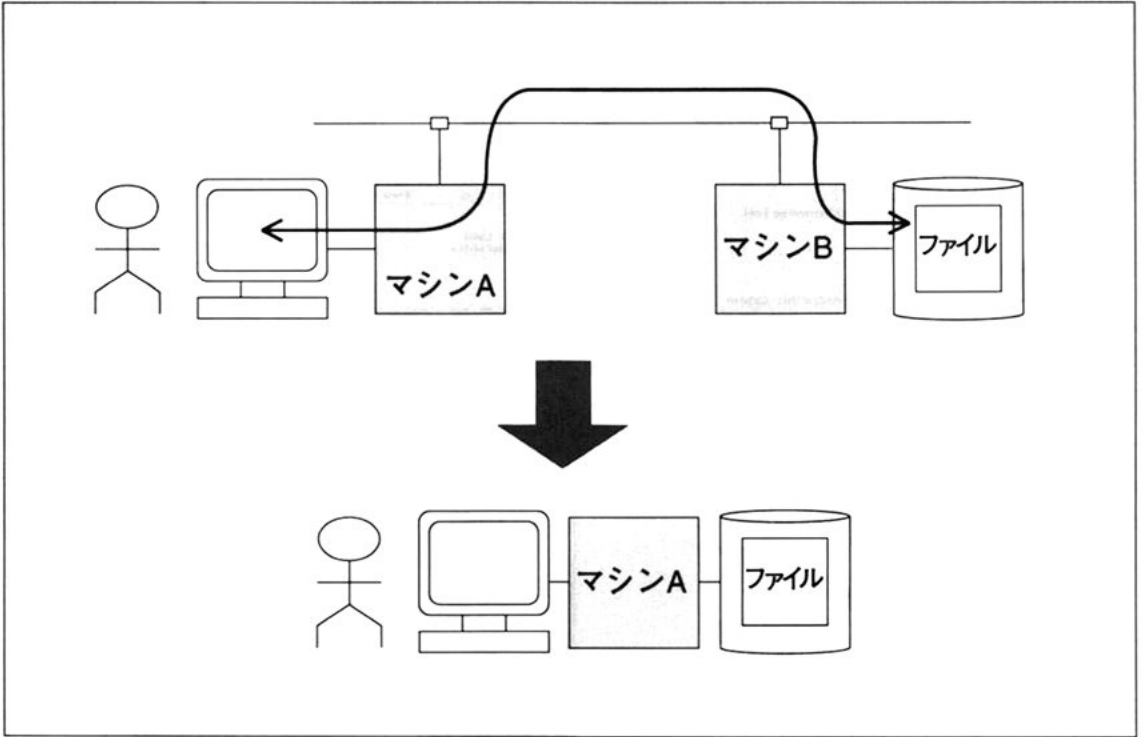
ファイル転送

や

ファイル共有

によって複数のユーザ間の情報（データ）の相互利用を可能にします。

UNIXでは、LANを利用するための機能もあらかじめ組み込まれています。

●LAN利用の概念

UNIXの利用形態

UNIXの利用形態としては、大きく分けて

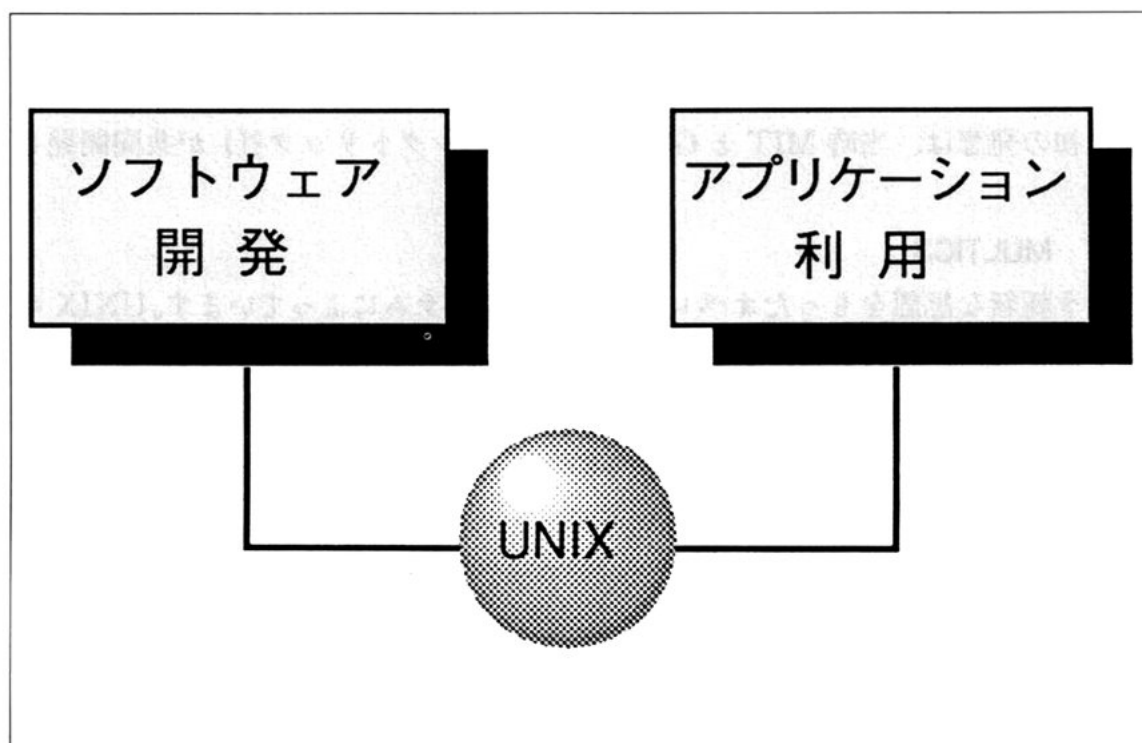
・ソフトウェア開発

・アプリケーション利用

があります。

UNIXはその誕生の経緯から開発志向性の強い機能を、特に研究者やソフトウェア開発者に提供してきました。ですから、ソフトウェア開発の効率を向上させるために十分な機能を備えているのは当然です。さらに、UNIX環境下で開発されたアプリケーションソフトウェアは、やはりUNIX環境下で利用するのが自然です。当初は、その開発志向性のために利用者にとっては必ずしも使いやすい環境であるとはいえませんでしたが、現在ではアプリケーションを利用する環境としてもX Window SystemをベースとしたGUI環境のおかげで、取り扱いやすいユーザインタフェース機能を備えるようになっています。

このような背景から、UNIX はエンジニアリング分野では CAD／CAM（コンピュータ支援による設計／製造）や AI（人工知能）などのシステムの OS として使われてきましたが、これに加えていわゆるビジネス分野のアプリケーションのためとしても使われるようになってきています。特に、DTP(Desk Top Publishing：卓上出版)や大規模なデータベースアプリケーションなどの利用には、パーソナルコンピュータでは不足な性能や機能をカバーする OS として重要な位置を占めるようになってきています。

1・2

UNIXの歴史と現状

1969年に、

AT&Tのベル研究所

で最初のUNIXが完成しました。今から約25年前のことです。UNIX開発への最初の発想は、当時MITとGE（ゼネラルエレクトリック社）が共同開発した

MULTICS

という斬新な思想をもったオペレーティングシステムによっています。UNIXの多くの特徴は、このシステムの思想からきているものだといわれています。

この後の何回かのバージョンアップによって、さまざまな開発者／研究者向きの機能を備えていきました。1973年の第5版で、UNIXは大きくその方向性を転換しています。それまでのUNIXは、それが動作するハードウェアの機能を最大限に発揮させるために、各ハードウェアに依存したアセンブリ言語を使って記述されていました。ところがハードウェア間の移植性や開発の容易性を重要視し、

C言語

を使って書き換えられたのです。C言語は、このときにUNIXを記述するために開発されたハードウェア依存の少ない高級言語です。現在では、UNIXがワークステーションをはじめとした多くのコンピュータの標準的なOSとなっていますが、C言語もUNIXばかりかそれ以外の多くのOS上でも標準的なプログラミング言語になっています。

1978年の第7版に至って、現在のUNIXの主な機能が整備されました。いわば、本来の意味でのUNIXの原型が完成したといえるでしょう。

さらに、この時期にAT&Tとは別に、カリフォルニア大学のバークレイ校で、本格的にUNIXを改造するプロジェクトが発足しました。つまり、UNIXにはAT&T版とバークレイ版（BSD：Berkley Softwear Distribution）という2

つの流れができたのです。

1989年になって、それぞれの本格的なバージョンが発表されました。つまり、

AT&TのSystemV

と

バークレイの4.2BSD

です。

System Vは、AT&TがUNIXをビジネスとして展開するための本格的なOSとして発表され、現在、UNIXベンダーのほとんどは、このSystemVをベースとした拡張版を提供しています。

一方、4.2BSDは特にネットワークの機能を強化し、ARPA(advanced reseach project agency：米国高等研究計画局）のプロジェクトとして開発が行われました。4.2BSDはOSの研究対象として使われるという意味で、商用のOSとしてのSystemVとは一線を画しています。

このような状況下で大きな問題が起きてきました。SystemVと4.2BSDという2つのUNIXの間に互換性がないことや、さらに同じSystemVをベースにした各社のUNIX間にもその拡張部分に非互換機能が増えはじめ、利用者の立場からすれば、同じUNIXでありながらその機能に関して互換性がないという大きな不都合が増えはじめたのです。

そこで、AT&Tと、4.2BSDを最も強力にサポートしているサンマイクロシステムズが協力し、相方の良い面を合流させた

SystemV rel.4.0

が開発されることになりました。これには比較的多くのメーカーが賛同しましたが、一方では、この動きに対抗してメインフレームの雄IBM、ミニコンの雄DEC、ワークステーションで積極的な展開をしているHPなどが

OSF(Open Softwear Foundation)

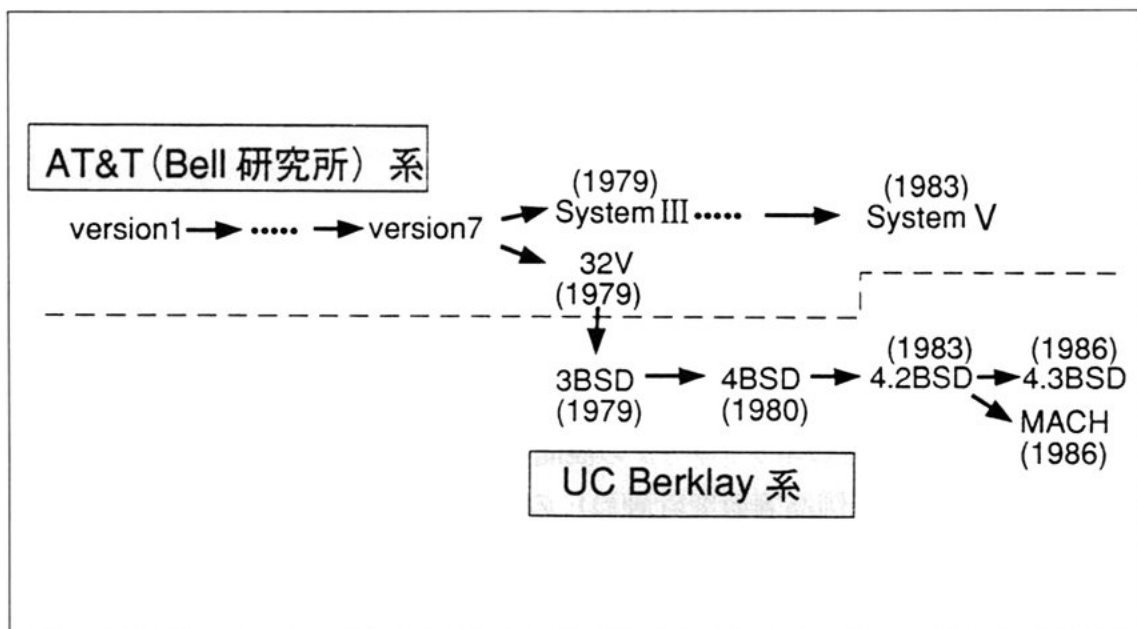
という団体を設立し、

OSF/1

というまた別のUNIXを提供しはじめました。

この数年間の対立の構図も、1993年にCOSEと呼ばれる両者が合流する団体が設立され、一本化の方向性が打ち出されています。

● UNIXの系譜

AT&T (Bell研究所) 系
version1 → ····· → version7
(1979)
System III ·····
(1983)
System V
32V
(1979)
3BSD
(1979)
4BSD
(1980)
(1983)
4.2BSD
(1986)
4.3BSD
MACH
(1986)
UC Berklay 系

1・3

基本操作と使用環境

開始から終了まで

UNIXは、同時に何人かのユーザが1つの環境を共同で使用することを前提としたマルチユーザ機能をもったOSです。このような環境では、使用できる資格をもたない部外者が、勝手にこのUNIXシステムを利用し、他人のファイルに対して改ざんや削除をしたり、悪意がなくても不可抗力で同様のダメージを与えてしまう可能性も考えられます。そのような事態を防ぐために、UNIXでは認められたユーザだけに利用を許可するような方式をとっています。

UNIXの利用を開始しようとするユーザは、毎回登録されているユーザかどうかをまずチェックされます。この手続きを

ログイン

と呼びます。

ログインは、ユーザがUNIXを使い始める際に、毎回必要な手続きです。ですから、すでにログイン名(ユーザ名)が、あらかじめUNIX側に登録されていなければなりません。ユーザ名が登録されていない場合は、

スーパーユーザ

と呼ばれるUNIXシステムの管理者に、ユーザとしての登録を依頼しなければなりません。

次頁の事例のように「login:」のプロンプトに続けてユーザ名(ここではfunamoto)を入力した後に、パスワードを入力します。これは、銀行のキャッシュカードなどを利用する際と同様の一種の暗証番号で、UNIXシステムは入力されたパスワードとシステムに登録されているパスワードを照合し、ユーザ本人であることを確認します。パスワードには、英文字・数字・特殊文字を使うことができます。なお、パスワードをキーボードから入力してもディスプレイには何も表示されません。これは、入力中のパスワードを他人に盗み見られないための

予防措置です。正しいユーザ名とパスワードが入力されると、開始メッセージが表示され、次にコマンド入力待ち状態を示すプロンプト

%

が表示されます。

パスワード名を間違えて入力した場合は、「incorrect」というエラーメッセージが表示されます。この場合は再びユーザ名の入力からやり直してください。

●ログインからプロンプトがでるまでの表示ログ（Sun & NEWS）

```
login : funamoto ⏎              ←――――ユーザ名の入力
Password : ______ ⏎             ←――――パスワードの入力
Last login : Fri Sep 6 10:11:11 from sun01   ←――――前回のログイン情報
SunOS Release 4.1.1-JLE1.1.1 (GENERIC) #1 : Mon May 21 13:46:13 JST 1990
%        ←――――――――――――――――――プロンプト

login : funamoto ⏎
Password:______ ⏎
Last login : Tue Nov 26 18:33:26 form nws01
NEWS-OS Release 4.0R # 0 : Tue Dec 18 19:02:18 JST 1990
%
```

なお、初めてログインする場合には、パスワードが設定されていないのが普通ですから、ユーザ名を入力するとすぐにプロンプトが現れ、UNIX を利用することができる状態になります。

パスワードの登録と変更(passwd)

パスワードの登録は、自分の環境を守るだけでなく、同じシステムを使っている多くのユーザの資産を不心得者から守るために絶対に必要な手続きですから、ぜひ行ってください。

まず、プロンプトに続けて passwd を入力します。パスワードの文字列を入力すると、再度確認のために入力を促すメッセージが出力されますから、ここでも同じ文字列を入力します。タイプミスなどで入力された 2 つの文字列が一致しないときには受理されません。再度、passwd コマンドを入力して、はじめからやり直します。

```
% passwd ⏎
Changing password for funamoto on sun01.
Old password:______ ⏎  ←――すでに登録されているパスワードを入力
New password:______ ⏎     (登録されていなければ表示されない)
Retype new password:______ ⏎
%
```

パスワードとして設定する文字列には、多少の制約があります。

・6文字以上の文字列（先頭から8文字までが有効）

・2個以上の英文字または1個以上の数字か特殊文字を含む

などの条件に合わなければなりません。これらの条件は、使用するUNIXのシステムによって多少異なりますからそれぞれのマニュアルで確認してください。

コマンドの入力

UNIXではプロンプトが表示されているときには、これに続けて

コマンド

を入力することによってさまざまな処理を行わせることができます。数多くのコマンドが用意されていて、これらを単独で利用するだけでなく、複数のコマンドを組み合わせて使うこともできます。

コマンドを入力するときの書式はほぼ定型化されています。すなわち、入力時のコマンド行は次の3つの部分からできています。

% コマンド名 [オプション][引数]

ここで[ ]は、ここの中の記述は省略してもよいことを示しています。オプションと引数については、おのおののコマンドが解釈しますから、その機能に応じて多少書式が異なります。そのためユーザは、引数としてどのようなものが指定できるかについては、大方のところは知っておく必要があります。

一般に、オプションではコマンドをどのように働かせるかを指示し、引数ではコマンドを働かせる対象を指示します。オプションは通常

-（ハイフン）

ではじまる文字列で指定します。例えば、引数に指定された文字列をそのまま出力する

echo コマンド

では、オプションの有無で次のように処理結果が異なります。

```
% echo "Hello!!" ⏎
Hello!!
%

% echo -n "Hello!!" ⏎
Hello!!%     ←――――――%が表示される前に改行されていない
```

echo コマンドの場合、

-n オプション

を指定すると、メッセージの出力後に改行をせずにプロンプトを表示します。また、オプションや引数を一般に省略した場合には、コマンド側では暗黙的な指定がなされているものと解釈してこれを実行します。この暗黙の指定を

デフォルト（default）

といいます。

例えば、この echo コマンドでは、オプションのデフォルトは

「文字列を出力した後、改行をする」

です。この例では、引数に指定されているのは、出力の対象となる文字列ですが、このほかに引数になるものの代表的なものとしては、ファイル名があります。詳しくは、1·4 節以降で説明します。

使用環境の設定（stty）

UNIX が柔軟性の高い OS であることは、ユーザが自分の使用する環境を好みに応じて変更することができる機能をもっていることでもわかります。代表的なものとして端末のキーに適当な機能を割り当てる

端末設定

と使用環境を自分の好みに設定する

個人環境設定

があります。

UNIX では、さまざまな種類のターミナル（端末）の使用を可能にするため

に、キーを端末の属性に合わせて設定することができます。これには、

stty コマンド

を使い、端末設定を行ったりその時点での設定状況を知ることができます。

```
% stty all ⏎
```

例えば、端末の機能キーを自分の好みに設定したいときには

```
% stty erase ^H kill ^U ⏎
```

のように指定します。ここでは1文字消去（erase）を CTRL + H に、また1行消去（kill）を CTRL + U に設定しています。

また、個人環境を設定するためには、ログインまたはログアウト時に自動的に実行されるファイルが利用されます。このファイルはコマンドインタプリタつまりシェルによって異なっています。Bシェルの場合には、ログイン時に自動的に実行される

.profile

に設定内容を記述します。一方、Cシェルではログイン時に実行される

.cshrc

.login

とログアウト時に実行される

.logout

があります。これらの3つのファイルの使い方については、1·5節のCシェルで詳しく説明します。

オンラインマニュアルの使い方（man）

UNIXには、数多くのコマンドのそれぞれに多彩なオプションがありますが、使用頻度の高いコマンドは別として、そのすべての使い方を覚えきれるものではありませんし、その必要もありません。例えば、複雑な処理をするために工夫したコマンド行を記述するときに、オプションや引数の詳細な指定方法を知りたいときには、マニュアルを参照すればよいのです。もちろん、書籍の形態になっているハードコピーマニュアルで構いませんが、UNIXでは、コマンド

やそのオプションの機能を端末からも確認することができる

オンラインマニュアル

を使うことができます。ハードコピーのマニュアルに比べて、その場で参照でき検索も速やかなので、その即応性からよく利用されます。

オンラインマニュアルを参照するには

man コマンド

を使います。例えば、lprm という印刷処理関係のコマンドの使用法を参照するときには、次のように入力します。

```
% man lprm⏎ ←———— lprmコマンドのマニュアルを参照する

LPRM(1)                  USER COMMANDS                  LPRM(1)

NAME
     lprm - remove jobs from the printer queue

SYNOPSIS
     lprm [ -Pprinter ] [ - ] [  job# ...  ] [  username ...  ]

DESCRIPTION
     lprm removes a job or jobs from a printer's spooling  queue.
     Since  the  spool  directory  is protected from users, using
     lprm is normally the only method by which a user can  remove
     a job.

     Without  any  arguments,  lprm  deletes  the  job  that   is
     currently  active,  provided  that the user who invoked lprm
     owns that job.

     When the super-user specifies a  username  lprm  removes  all
     jobs belonging to that user.

 --More--(21%)←———— スペースキーの入力
     You can remove a specific job by supplying its job number as
     argument, which you can ob                           exam-
```

この表示内容は、ハードコピーのマニュアルとまったく同じ内容です。

また、コマンド名自身がわからないときに、行いたい作業に関連したキーワードから、コマンドを検索することもできます。

```
% man -k editor⏎ ←——editor というキーワードを指定してマニュアルを検索する
COFF (5)              - common assembler and link editor output
a.out (5)             - assembler and link editor output format
ed (1)                - basic line editor
ex, edit, e (1)       - line editor
fontedit (1)          - a vfont screen-font editor
ld, ld.so (1)         - link editor, dynamic link editor
ldconfig (8)          - link-editor configuration
link (5)              - link editor interfaces
sed (1V)              - stream editor
textedit (1)          - SunView window- and mouse-based text editor
vi, view (1)          - visual display editor based on ex(1)
%    ↑                    ↑
   コマンド名              解説
```

ここでは、man コマンドに -k オプションを指定してエディタ(editor)に関連したコマンドの名前と機能を一覧にして表示させています。これでコマンド名さえわかれば、あとはオプション指定なしの man コマンドの引数にコマンド名を指定して詳細な使用方法を表示させます。

ログアウト(logout)

UNIX のコマンドやアプリケーションソフトウェアを使って一連の作業が終了した後、コマンド入力待ちのプロンプトが出ている状態やアプリケーションが立ち上がっている状態で端末やワークステーションを放置してはいけません。放置したままにすると、他人に無断で使われる可能性があります。

これでは、せっかくパスワードを設定してユーザの資源を保護している意味がなくなってしまいます。

このような理由で、UNIX での作業が終了したら、必ず UNIX から抜け出すために

ログアウト

という操作を行わなければなりません。そのうえで、端末の場合には電源を切断するのです。なお、ワークステーションの場合には、スーパーユーザの資格をもった人が、電源を切る前に必要なシャットダウンという操作を行います。

ログアウトを行うには、

[CTRL]＋[D]　または　logout ⏎

を入力します。

```
% logout ⏎


login:
```

ログアウト行うと自動的に回線が切断され、「login:」が表示されて再びログイン待ちの状態に戻ります。

このように、UNIX システムに対してログインを行い、ユーザがコマンドを使ってさまざまな要求を出し、これに対しシステムが処理サービスを行い、最終的にはログアウトをするという、ユーザとシステムとの会話の一連の繰り返しを

セッション

といいます。

1・4

ファイルシステム

ファイルとその操作

コンピュータ上で取り扱われるデータの集合は、ディスク装置などの補助記憶装置上に保存するときには、

ファイル（file）

という単位で管理されます。ファイルには、それぞれ固有の名前を付けることができますから、この名前を使ってひとまとまりの文書や社員の履歴データというような、特定の意味をもつデータの集合を簡単に取り扱うことができます。SystemV 系の UNIX では 14 文字、4BSD 系では 255 文字以内で任意の長さの文字列を名前として付けることができます。

UNIX のファイルの特徴は、

特に定まった形式がない

というところにあります。すべてのユーザのファイルは、

1バイト単位のデータ

が集合した形式でできています。ですから、UNIX にはファイルを取り扱うコマンドは、どのようなファイルに対してもほぼ共通に使えます。

UNIX の管理するファイルには、大きく 3 種類のファイルがありますが、このうち

通常ファイル（ordinary file）

は、プログラムのソースリストやワープロの文書などのデータを格納するテキストファイルやプログラムオブジェクトの実行ファイルなどの典型的なファイルのことです。

特に、テキストファイルを操作するコマンドは数多くあります。ファイルの内容を表示する

cat コマンド

ファイルの先頭部分や末尾部分を出力する

head コマンド

tail コマンド

ファイルの内容を並べ替える

sort コマンド

ファイルを複写、移動、削除する

cp コマンド

mv コマンド

rm コマンド

などが代表的なものです。

●ファイルの種類

通常ファイル	利用者のデータを一まとまりにして名前を付けたもの プログラムソース、プログラムオブジェクト ドキュメント
ディレクトリファイル	ファイル管理用のファイル 他のファイルやディレクトリへの連結関係をもっている
特殊ファイル	入出力装置をアクセスするためのファイル /dev ディレクトリの下にある

●コマンドの実行例(テキストファイルの操作)

```
% cat exfile1 ⏎ ←———— ファイルの内容を表示する
Funamoto Susumu     Tokyo
Date Masamune       Sendaii
Takeda Shingen      kofu
%

% cat -n exfile1 ⏎ ←———— ファイルの内容を行番号を付けて表示する
1  Funamoto Susumu     Tokyo
2  Date Masamune       Sendai
3  Takeda Shingen      Kofu
%
```

```
% head -6 exprog.c ⏎ ←──── ファイルの先頭6行を表示する
/*
   Filter Program Example1
*/
#include <stdio.h>
main()
{

% tail exprog.c ⏎ ←──── ファイルの末尾部分10行を表示する
                y = y * i;
        }
        return(y);
}
err()
{
        printf("error");
        exit(0);
}
/* Filter program end */
%
```

```
% cat exfile1 ⏎ ←──── ファイルの内容を表示する
Funamoto Susumu    Tokyo
Date Masamune      Sendaii
Takeda Shingen     kofu
% cp exfile1 exfile1.bak ⏎ ←── ファイルの複製をつくる
% ls ⏎ ←──── exfile1.bak が作成されていることを確認
exfile1         exfile2         exprog.c
exfile1.bak     exfile3
% cat exfile1.bak ⏎ ←──── exfile1.bak の内容を表示する
Funamoto Susumu    Tokyo   ┐
Date Masamune      Sendaii ├←── exfile1 と同一の内容
Takeda Shingen     kofu    ┘
%
```

```
% ls⏎ ←────── ディレクトリ内容を表示する
exfile1  exfile2  exprog.c
% mv exprog.c exfile3⏎ ←── ファイルの名称を変更する
% ls⏎
exfile1  exfile2  exfile3
% rm exfile3⏎ ←────── ファイルを削除する
% ls⏎
exfile1  exfile2
%

% cat exlist.dat⏎ ←──── exlist.bat の内容を表示する
Funamoto Susumu    Tokyo
Date Masamune      Sendai
Takeda Shingen     Kofu
Funamoto Syotaro   Tokyo
Date Tadamune      Sendai
Takeda Katsuyori   Suwa
% sort exlist.dat⏎ ←─── exlist.bat の内容を並べ替える
Date Masamune      Sendai
Date Tadamune      Sendai
Funamoto Susumu    Tokyo
Funamoto Syotaro   Tokyo
Takeda Katsuyori   Suwa
Takeda Shingen     Kofu
%
```

また

ディレクトリファイル（directory file）

は、ファイルを能率よく管理するために、他のファイルやディレクトリへの連結関係をもっているファイルです。単にディレクトリと呼びます。このディレクトリの構造を使って、多数のファイルを機能的に管理できます。このようなファイルの集まり全体のことを

ファイルシステム

と呼んでいます。

このほかには、

特殊ファイル(special file)

と呼ばれるものがあります。これはUNIXの代表的な特徴の一つにもなってい

るのですが、ハードディスクやディスプレイなどといった入出力装置をファイルと同様の操作で扱えるようにするために必要なファイルです。端末装置、ディスク装置、テープ装置などへの入出力が、ディレクトリの

/dev

にあるファイルを通して共通のファイルと同様に操作できます。

例えば、コマンドの実行結果をファイルに書き込むリダイレクション機能を使ってファイルの内容を複写するときに、相手がディスク上のファイルでも、特殊ファイル経由の端末であっても、次のように同一の形式でコマンドを実行することができます。

```
% cat file > newfile ⏎    ←——file の内容をディスク上のファイル newfile に出力する
% cat file > /dev/tty10 ⏎ ←——file の内容を端末 tty10 に出力する
```

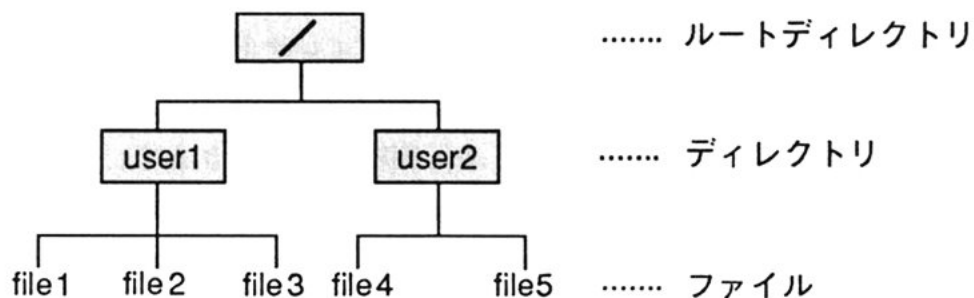
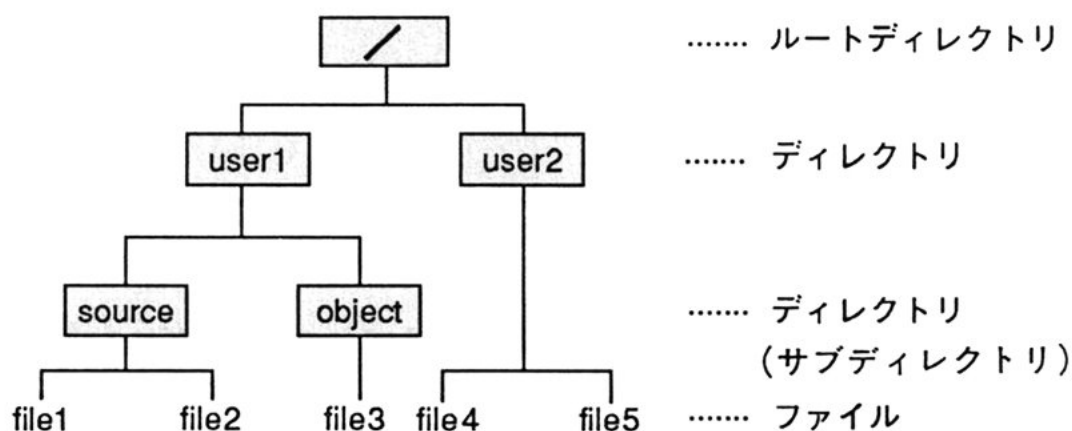
ディレクトリとその操作

ファイルの数が増えてくれば、それぞれのファイルにいろいろ工夫した名前を付けても、欲しいファイルを探し出したり、名前だけからファイルの内容を連想することが難しくなってきます。

そこで、UNIX では

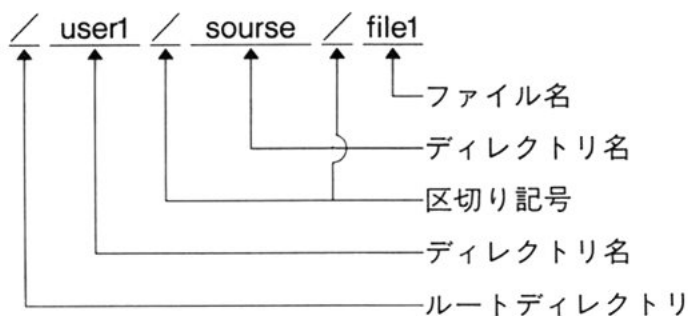
ディレクトリ

と呼ばれる特別なファイルをグループ別に用意して、この中にグループに属するファイルの名前を記述し、ファイルを管理することができるようになっています。ディレクトリによって管理されたファイルは、ツリー構造を形成します。これで、ファイルを分類して整理することができるようになります。

例えば、2 人のユーザがいて、それぞれのファイルを分けて管理するためには、2 人のそれぞれのディレクトリ user1、user2 の下に各自のファイルを置きます。

ディレクトリは、ディレクトリ自身をも管理できますから、例えば、ユーザ1が自分のファイルをさらに分類することもできます。プログラムのソースファイル file1 と file2 をディレクトリ source に、オブジェクトファイル file3 をディレクトリ object に配置すれば、次図のようになります。

／は、

ルートディレクトリ

といい、UNIX のファイルシステムの大基になっています。このルートディレクトリを起点とすれば、トップダウン的にすべてのファイルをたどることができます。

この場合に、file1 を参照するための記述は次のとおりです。

これは、file1にたどり着くまでの経路をルートディレクトリを起点として、その経路にあるディレクトリを順番に「/」で区切って表記しています。この経路のことを

パス(path)

と呼んでいます。ここでは、/user1/source/の部分がパス名です。特にこのようにルートディレクトリ「/」からの指定をするパスを

絶対パス

と呼んでいます。そこで、例えばこのファイルの内容を表示させたければ、次のようなコマンドを入力します。

```
% cat /user1/source/file1 ↵
```

このようにファイルシステムの中にあるファイルを指定するには、ファイル名の先頭にパス名を付けるのが基本です。ところが、木構造の階層の深いところにあるファイルの場合には、このパス名が非常に長くなってしまいます。例えば

/user1/source/c/version1/file1.c

のような長いパス名で修飾されたファイル名を毎回コマンド行に記述するのは面倒です。

そこで、

ワーキングディレクトリ(またはカレントディレクトリ)

という便利な概念があります。ワーキングディレクトリは、デフォルトで指定されているパス名のことです。つまり、何もパス名を指定しないファイル名や、ルートディレクトリが記述されていないパス名とファイル名の記述では、それぞれの先頭に暗黙的に指定されているのと同じに取り扱われるパス名のことです。

いま、ワーキングディレクトリが

/user1/source

であれば、/user1/source/file1を指定するには、この/user1/sourceの部分を省略してfile1だけを指定すればよいのです。このように、ワーキングディレクトリを起点としたパスの指定の仕方を

相対パス

指定と呼びます。ワーキングディレクトリは、ログイン時に各ユーザごとに自動的に設定されます。これを特に

ホームディレクトリ

と呼んでいます。なお、現在のワーキングディレクトリを知るためには、

pwd コマンド

を使います。

```
% pwd ⏎
/user1/source
%
```

この設定状態では、単にfile1と/user1/source/file1は同じファイルを意味することになりますから、file1だけを記述すればよいわけです。

ワーキングディレクトリは、ユーザが自ら

cd コマンド

を使って変更することができます。

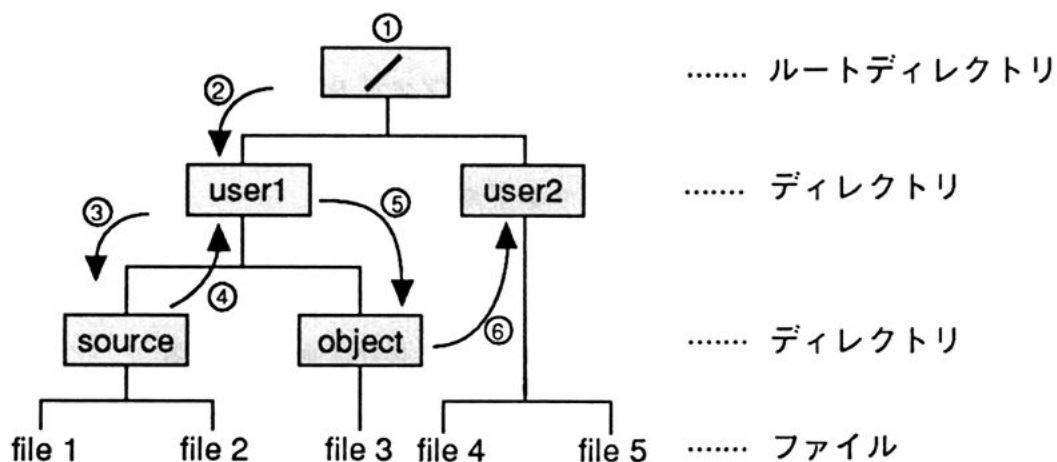
```
% cd /usr1/object ⏎
%
```

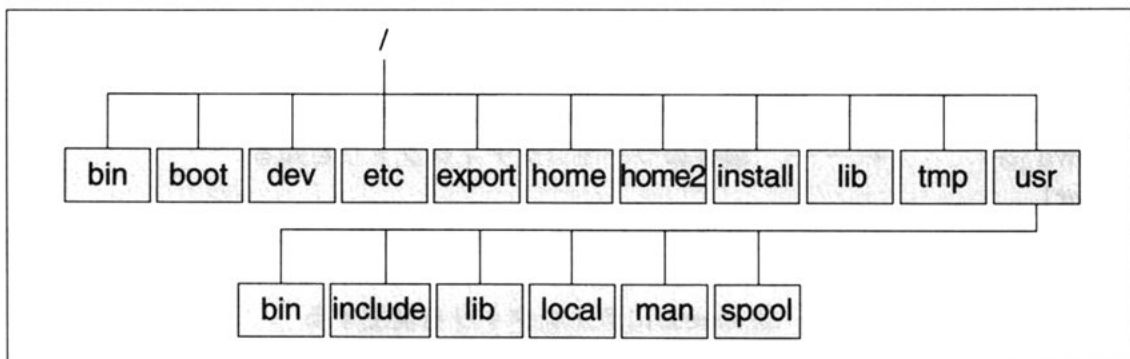
```
% pwd ⏎
/user1/object
%
```

ワーキンディレクトリに、どのようなファイルが存在するかを知るためには、ディレクトリの内容を表示する

ls コマンド

を使います。以上のディレクトリコマンドを使ってルートディレクトリから出発して、ファイルシステムの中をすべて検索する例を次頁の図に示します。

```
% cd  /⏎         ←─── ①ディレクトリをルートディレクトリに変更する
% ls ⏎           ←───   ルートディレクトリの内容を表示させる
user1   user2    ←───   2つのサブディレクトリを表示
% cd user1 ⏎     ←─── ② user1 にディレクトリを変更する
% pwd ⏎          ←───   現在のワーキングディレクトリを知る
/user1
% ls ⏎           ←───   サブディレクトリ user1 の内容を表示させる
source object
% cd source ⏎    ←─── ③ source にディレクトリを変更する
% ls ⏎
file1 file2
% cd .. ⏎        ←─── ④ source の親ディレクトリに変更する
% pwd ⏎
/user1
% cd object ⏎    ←─── ⑤ object にディレクトリを変更する
% ls ⏎
file3
% cd /user2 ⏎    ←─── ⑥ user2 にディレクトリを変更する
% ls ⏎
file4 file5
%
```

参考までに SystemV 系の UNIX のファイルシステムの構造は次図のとおりです。他の UNIX の場合でも大方は同じような構造をしています。

●ファイルシステムの例

/bin　：実行形式のコマンドが存在する
　　　(/usr/bin にシンボリックリンクされている)
/boot　：ブート時に必要なファイルが存在する
/dev　：特殊ファイルが存在する
/etc　：システム固有の管理用ファイルが存在する
/export：ネットワークにエキスポートされる
/home　：ユーザのホームデイレクトリを置く
/home2：ユーザのホームデイレクトリを置く
/install：追加パッケージを置く
/lib　：言語用のライブラリファイルが存在する
　　　(/usr/lib にシンボリックリンクされている)
/tmp　：テンポラリー（一時的作成）ファイルをつくる
/usr　：例えば下記のディレクトリが存在する
　/usr/bin　：実行形式のコマンドが存在する
　/usr/include：include ファイルやヘッダファイルが存在する
　/usr/lib　：ユーティリティで使うライブラリファイルが存在する
　/usr/local　：システムで必要なコマンドなどが存在する
　/usr/man　：オンラインマニュアルが存在する
　/usr/spool　：スプールのファイルを置く

ディレクトリはユーザが必要に応じて作成したり、不要なものを削除することができます。ファイルシステムの中にディレクトリを作成するには、

　mkdir コマンド

を使います。 また、ディレクトリを削除するには、

rmdir コマンド

を使います。ただし、この際に削除するディレクトリの下にはファイルやディレクトリがあってはいけません。

```
% pwd ⏎
/usr/user1
% ls ⏎
% mkdir dir1 ⏎          ←―――― user1 の下にディレクトリ dir1 を作成する
% ls -l ⏎
% cd dir1 ⏎             ←―――― dir1 にディレクトリを変更する
% pwd ⏎
/usr/user1/dir1         ←―――― ワーキングディレクトリは dir1 にある
%
```

```
% cd .. ⏎               ←―――― dir1 の親ディレクトリ user1 に変更する
% pwd ⏎
/usr/user1              ←―――― ワーキングディレクトリは user1 にある
% ls ⏎
dir1
% rmdir dir1 ⏎          ←―――― ディレクトリ dir1 を削除する
% ls ⏎
%                       ←―――― ディレクトリ dir1 は削除された
```

ところで ls コマンドは、単にファイル名を表示するだけでなく、ファイルの容量、作成日付やファイル名などの詳細な情報をも表示することができます。

これには、

-l オプション

を指定します。

```
% ls ⏎
user1   user2
% ls -l ⏎
total 29
drwxr-xr-x   2  funamoto     512 Jun 22 05:31   user1
drwxr-xr-x   2  funamoto     512 Jun 29 22:42   user2
% cd user1 ⏎     ←―――― user1 にディレクトリを変更する
% ls -l ⏎
total 29
```

```
-rwxr-xr-x   2   funamoto      512  Jun  22  05:31   file1
-rwxr-xr-x   2   funamoto      512  Jun  29  22:42   file2
%
```

一番左側に表示されるモードは、ファイルの種類とそのファイルに対するアクセスの資格を一連の文字列で表示したものです。各ファイルに対して所有者自身、自分が属しているグループ、そして全ユーザという3つのレベルに分けてこのファイルを利用できるかどうかの範囲を指定できます。

```
－  rwx  rwx  rwx
↑    ↑    ↑    ↑
①   ②   ③   ④
```

①	ファイルの種類	- 通常ファイル d ディレクトリファイル
②	所有者に対する保護	r 読み出しを許可 w 書き込みを許可 x 実行を許可 — 不許可
③	グループに対する保護	
④	全ユーザに対する保護	

例えば、file1の保護モードは

-rwxr-xr-x

となっています。まず最初の1文字が

-

ですから

通常のファイル

で、次に続く自分に対する保護モードを表す3文字はすべてが有効ですから

自分だけが読み書き実行可能

で、残りの2組の3文字は

r と x

だけが有効となっていますから

グループやその他のユーザに対しては読み出しと実行だけが許可

されていることがわかります。

なお、保護モードはユーザが自ら設定することができます。これには、

chmod コマンド

を使います。

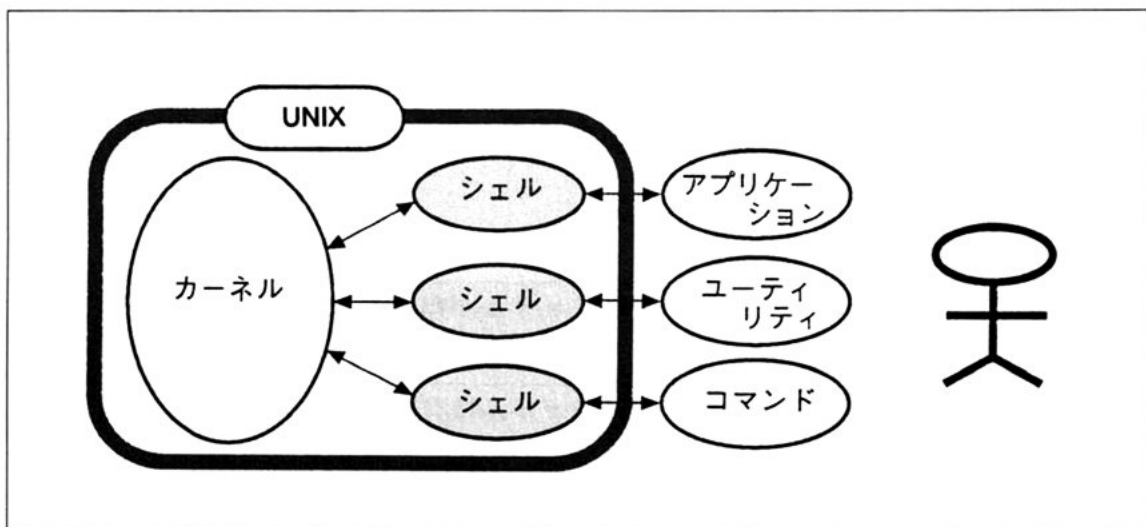
1・5

Cシェル

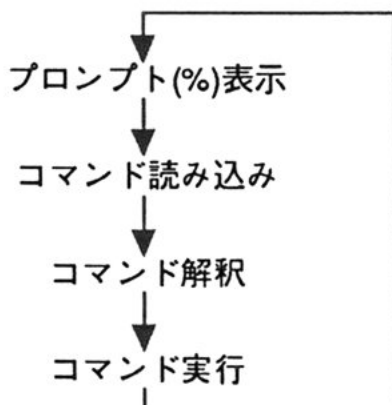
シェルは、「殻」という意味で、文字どおりUNIXの最も外側にあるソフトウェアです。UNIXを使う際のユーザインタフェース部分だともいえます。つまり、UNIX基本部分とユーザの仲介をしてコマンド入力の便宜を図ってくれる

コマンドインタプリタ

の役割をはたしてくれるUNIXのサブシステムです。

●Cシェル概念図

シェルの大きな特徴の一つは、ユーザが思いどおりにその利用環境をカスタマイズできる点にあります。決まりきった作法にユーザが合わせるのではなく、ユーザが自分の作法で快適にUNIXを利用できるように変更することができるのです。

UNIXで普通コマンドを実行させるときには、プロンプト「%」に続けてコマンド行を入力しますが、これがすなわちシェルを利用することそのものなのです。この際に、シェルは入力されたコマンドを読み込み、これを解釈してさらに必要な処理を実行する役割を一手に引き受けています。

●Cシェルの処理手順

なお、コマンドには表に示すとおり、実行ファイルそのものが存在するコマンドのほかにも、シェルに組み込まれたコマンドやシェルが記述内容を解釈して実行するシェルスクリプトがありますので、それぞれ用途や形式を区別して覚えておいてください。

●実行コマンドの種類

コマンド・オブジェクトファイル	実行ファイルが存在する(オブジェクト形式)
シェルスクリプトファイル	実行ファイルが存在する(テキスト形式)
シェル組込みコマンド	実行ファイルは存在しない

シェルにはさまざまな種類がありますが、現在最も一般的に使われているシェルには、AT&TのベルΒ研究所のSteven Bourneによって開発され、初期のUNIXから標準添付されていてる

　　Bシェル　(Buorneシェル)

と、バークレイ版のUNIXに添付されて提供される

　　Cシェル

があります。

Cシェルには、C言語風な構文が採用されていていますから、Cプログラマにとっては、なじみやすいツールです。また、以前に実行したコマンドを再利用する

ヒストリ

や複数のジョブの実行を制御する

ジョブ制御

などの便利な機能があります。

最近では、Cシェルは AT&T 版をベースとした UNIX でも、ほぼ例外なく採用されていますので、特に対話的な処理を必要とされるときには、一般的に利用されます。ここでは、主にCシェルの機能を説明します。

Cシェルの起動と終了

Cシェルを立ち上げる一つの方法は

csh コマンド

を入力することです。ところが、実際にはユーザがログインをしたときにCシェルは自動的に起動されています。これは、利用者を管理しているパスワードファイル

/etc/passwd

に、ログイン時の起動シェルとしてCシェルの起動の指定がされているからです。

```
% cat /etc/passwd ↵
 ⁀
funamoto:■■■■:102:200:S.Funamoto:/usr/users/funamoto:/bin/csh
 ⁀                                                        ↑
%                                                 Cシェル起動の指定
```

ですからログインが完了すればCシェルのプロンプトの「%」が表示されてコマンド入力待ち状態になるわけです。

起動シェルを例えばBシェルに変更したいときには、

chsh コマンド

を使います。

```
% chsh funamoto /bin/sh ↵
```

この例では、ユーザ名funamotoのログイン時に立ち上げるシェルをBシェルに変更しています。/etc/passwdファイル中に記述されている文字列/bin/cshはCシェルですし、/bin/shはBシェルのファイルです。

Cシェルを終了するときには

　　CTRL + D　または　logout ↵

を入力します。

ジョブの制御

シェルを経由して入力される一つのコマンド行は、ひとかたまりの処理単位として扱われます。これを

　　ジョブ

と呼んでいます。例えば、複数のコマンドをパイプを使って入力したときにはこれ全体が一つのジョブとして

　　ジョブ番号

が付けられて管理されます。一方、一つのコマンド行でもそれぞれのコマンドは異なったプロセスとして実行されますから、それぞれ異なった

　　プロセスID

が付けられます。コマンドを実行中に必要に応じた制御を行うときには、このジョブまたはプロセスを単位とします。

ジョブ番号は各シェルごとの通し番号ですが、プロセス番号はシステム全体の通し番号で管理されます。

このようなプロセスの状態を確認するためには

　　psコマンド

を使います。

```
% ps ⏎ ◀──────────── プロセスの状態を確認する
  PID TT  STAT  TIME  COMMAND
  765 p0  S     0:02  -csh (csh)   ◀──── Cシェルのプロセス
  919 p0  R     0:00  ps           ◀──── ps コマンドのプロセス
%
```

コマンドを入力してジョブを実行する場合には、このジョブが端末を専有します。このジョブを

フォアグラウンドジョブ

といいます。これでは、同時に別の作業ができなくなりますから、

バックグラウンドジョブ

として実行し、別のコマンドの入力をできるようにすることもできます。

```
% cc exfile.c & ⏎ ◀──── exfile.c のコンパイルをバックグラウンドで実行する
[1]  916
% ps ⏎            ◀──── プロセスの状態を確認する
  PID TT  STAT  TIME  COMMAND
  765 p0  S     0:02  -csh (csh)
  916 p0  S     0:00  cc exfile.c                ┐ コンパイルのプロセスと同時にps
  918 p0  R     0:00  ccom - -fsoft -mc68020     ┘ コマンドのプロセスが実行される
  919 p0  R     0:00  ps
%
```

ジョブをバックグラウンドで実行させれば、シェルのプロンプトは即座に表示されますから、別のジョブを実行させることができます。バックグラウンドジョブとして実行させるには、コマンド行の最後尾に

&

を付けます。

```
% find / -name core -print > outfile & ⏎
[1] 192
%
```

find コマンドは、指定された名前のファイル (-name core) を、引数として記述されたディレクトリ（/ はルートディレクトリ）以下のすべてのファイルの中から探し出して、この結果を出力（-print）してくれます。

このようにして実行させたジョブの状態を確認するには、

jobs コマンド

を使います。

```
% cc exfile.c & ⏎ ←———— findコマンドに続けてccコマンドを実行する
[2] 198
% jobs ⏎
[1]  + Running        find / -name core -print > outfile  } 同時に2つのジョブ
[2]  - Running        cc exfile.c                          } が実行されている
%
```

ジョブの状態を示すキーワードとして

Running ：実行
Stopped ：一時停止
Terminated ：強制終了
Done ：正常終了
Exit ：異常終了

などが表示されます。

フォアグラウンドで実行されたジョブを一時停止状態にするには、

[CTRL] + [Z]

をキーボードから入力します。

```
% cc   exfile.c ⏎
^Z    ←————————— [CTRL]を押しながら[Z]を押す
Stopped
% jobs -l ⏎
[1]  +  868 Stopped                cc exfile.c
%
```

この状態から再び実行を再開させることができます。フォアグラウンドのジョブとして実行させるには

fg コマンド

を入力し、バックグラウンドのジョブとしては

bg コマンド

を入力します。

この一連の操作でフォアグラウンドジョブに対して一時停止状態を経由して、バックグラウンドジョブに変更して実行させることができることがわかります。つまり、CTRL ＋ Z を入力したあとでbgコマンドを入力すればよいわけです。

なお、バックグラウンドで実行状態にあるジョブを一時停止状態にするには、

stop コマンド

を使います。

```
% jobs ⏎
[1]     Running                 cc exfile.c
% stop ⏎
[1]+ Stopped (signal)           cc exfile.c
%
```

なお、ジョブを完全に終了させてしまうには、

kill コマンド

を使います。引数にはジョブ番号を指定します。オプションなしでは、完全に終了しない場合には

-9 オプション

を指定してください。

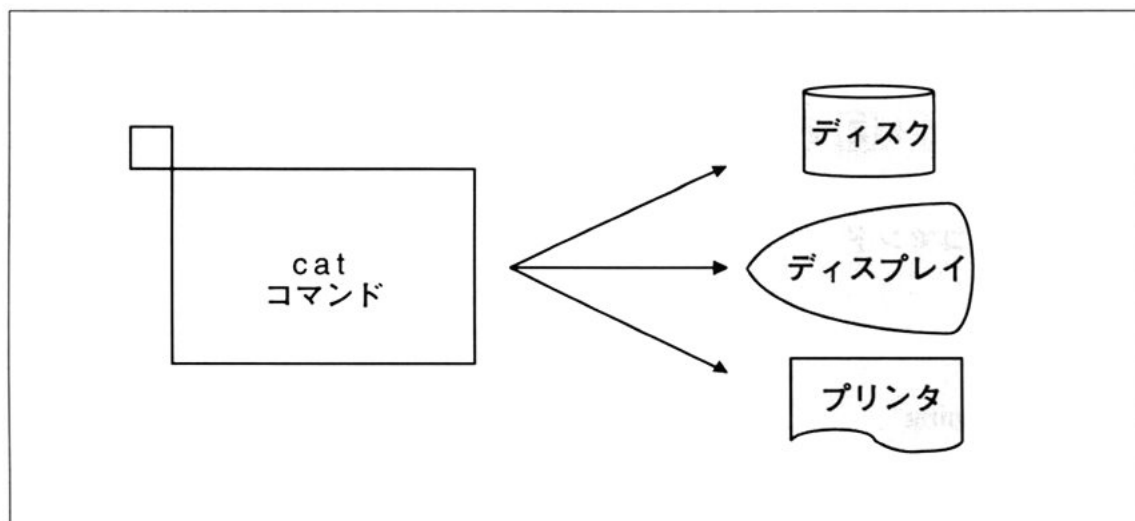
```
% kill 1 ⏎
```

なお、フォアグラウンドで実行されているジョブを強制終了させるときには

CTRL ＋ C

を入力して、ジョブに割り込みを通知します。

標準入出力

UNIXでは、すべての入出力の対象をファイルとして統一的に取り扱っています。つまり、ディスク上にあるいわゆるファイルだけでなく、キーボード、ディスプレイをはじめとしたあらゆる入出力装置がファイルとして扱われます。このようにすることによって、各装置ごとに別々のコマンドを使わずに、共通のコマンドで操作ができるようになります。

ところで、すべてのコマンドを入力する際に明示的に入出力のファイルを指定するのは、煩わしいものです。そこで暗黙の指定すなわちデフォルト値が採用されるようになっています。

入力のファイルとして

標準入力（stdin）

出力ファイルとして

標準出力（stdout）

エラーメッセージのファイルとして

標準エラー出力（stderr）

が用意されています。

これらの標準ファイルは、通常は端末装置に割り当てられていますから、標準入力がキーボード、標準出力と標準エラー出力がディスプレイになっています。

ですから、一般的にコマンドを実行するときには、入力データはキーボードから入力され、処理結果がディスプレイに表示されるのです。

catコマンドは、引数にファイル名を指定すれば、これを入力としてこの内容を表示しますが、上の例では指定がありません。そのため標準入力（キーボード）からのデータ入力を待ちます。キーボードからa、b、cが入力されれば、そのつどcatコマンドはこの内容を標準出力（ディスプレイ）に出力します。

コマンド入力行の構造

UNIXコマンド入力の方法は、基本的には前述のとおり、

コマンド名［オプション］［引数］

です。Cシェルでは、特殊文字を使うことによって次表のとおり、さらに柔軟なコマンド入力をすることができます。

●コマンド入力のバリエーション

<table>
<tr><th>機　能</th><th>書　式</th></tr>
<tr><td>連続実行</td><td>% cmd1; cmd2; … ; cmdn; …</td></tr>
<tr><td>並列実行</td><td>% cmd&</td></tr>
<tr><td>パイプライン</td><td>% cmd1 | cmd2 | … | cmdn | …</td></tr>
<tr><td>リダイレクション</td><td>%cmd <入力ファイル名</td></tr>
<tr><td></td><td>%cmd >出力ファイル名</td></tr>
<tr><td>(追加)</td><td>%cmd >>出力ファイル名</td></tr>
<tr><td>グループ化</td><td>%(cmd1; cmd2; … ; cmdn; …)</td></tr>
<tr><td>条件実行</td><td>% cmd1 &cmd2</td></tr>
<tr><td></td><td>% cmd1 ‖ cmd2</td></tr>
</table>

cmd はコマンドを表す

■連続実行

1回のコマンド行の入力で複数のコマンドを連続して実行させることができます。

```
% mkdir dir1;cd dir1;vi file1 ⏎
```

この例では、カレントディレクトリに新たにディレクトリ（dir1）を作成し、そこへカレントディレクトリを移し、新たなファイル名（file1）を指定して vi エディタを使用するという3つのコマンドをセミコロン「；」で区切って1回の入力だけで実行させています。

■並列実行

UNIX では、同時にいくつかのジョブを実行させることができます。通常のコマンド入力では、端末からの入力を専有するフォアグラウンドジョブとして実行されますから、このコマンドが終了しなければ他のコマンドを入力できません。

そこで、他のコマンド入力を可能にしたまま、別のコマンドをバックグラウンドジョブにして同時に実行します。

```
% cc file.c & ⏎
[1]  1234
% ps ⏎              ←―――― コンパイルジョブが終了する前に入力できる
```

この例では、まずC言語のソースファイル（file1.c）のコンパイルをバックグラウンドで実行します。このとき「%」のプロンプトが即座に返却されてきますから、ps コマンドを入力できます。この際、ps コマンドはフォアグラウンド

ジョブとして実行されます。

■パイプライン

ほとんどのUNIXのコマンドは、標準入力からデータを受け取って標準出力に結果を出力しています。この機能を果たすコマンドを

　　フィルタ

と呼びます。そこで、複数のコマンドの間で直続的にデータを受け渡して、複合した処理をさせたいときに、

　　パイプライン

を使って、あるコマンドの標準出力と他のコマンドの標準入力を直接結合して連続的な処理を行います。

```
% ls -l | sort -n +3 ⏎ ←───── ディレクトリリストをファイルの
total 226                        容量の昇順にソートして表示する
-rw-r--r--  1 funamoto       27 Nov 13 16:57  xab
-rw-r--r--  1 funamoto       27 Nov 13 16:58  outfileab
-rw-r--r--  1 funamoto       30 Oct 28 10:27  exfile.dat
-rw-r--r--  1 funamoto       40 Nov 13 16:57  xaa
-rw-r--r--  1 funamoto       40 Nov 13 16:58  outfileaa
-rw-r--r--  1 funamoto       44 Oct 28 10:24  exfile2.txt
-rw-r--r--  1 funamoto       58 Oct 28 10:18  exfile1.txt
-rw-r--r--  1 funamoto       58 Oct 28 15:15  exfile.txt
-rw-r--r--  1 funamoto       67 Oct 19 15:57  students.dat
-rwxr-xr-x  1 funamoto       67 Oct 29 10:32  infile
-rw-r--r--  1 funamoto      282 Oct 16 18:33  exfile2.c
-rw-r--r--  1 funamoto      290 Oct 24 16:34  exfile1.c
-rw-r--r--  1 funamoto      296 Nov  1 13:04  exfile.c
-rw-r--r--  1 funamoto      453 Oct 19 15:59  total.awk
drwxr-xr-x  2 funamoto      512 Oct 24 16:18  mk
drwxr-xr-x  2 funamoto      512 Oct 29 12:36  link
drwxr-xr-x  3 funamoto      512 Oct 16 18:36  cc
-rw-r--r--  1 funamoto      989 Oct 29 10:22  exfile1.o
-rwxr-xr-x  1 funamoto    24576 Oct 16 14:22  exfile1
-rwxr-xr-x  1 funamoto    24576 Oct 28 15:45  exfile
-rwxr-xr-x  1 funamoto    24576 Oct 29 14:52  a.out
-rw-r--r--  1 funamoto  2188090 Oct 16 14:28  core
%                        └──── ファイルの容量
```

この例では、lsコマンドで出力されるディレクトリの情報をsortコマンドに渡して昇順に並べ直して出力します。このときに中間的なファイルは作られません。

■リダイレクション

コマンドの入出力先のデフォルトは、端末になっています。ですから、通常の場合にはコマンドの標準入力は端末に接続されたキーボードですし、標準出力と標準エラー出力は同様にディスプレイ画面になっているのです。この標準入出力が割り当てられている先を端末からファイルに変更する方法として

リダイレクション

があります。

```
% cat file1.dat ⏎
3
2
1
% sort <file1.dat >file2.dat ⏎  ←── file1.dat を読み込み、ソート
                                     file2.dat に書き出す
% cat file2.dat>>file1.dat ⏎    ←── file2.dat の内容を file1.dat に追加する
% cat file1.dat ⏎
3
2
1
1 ┐
2 ├ file2.dat から追加された内容
3 ┘
%
```

catコマンドは、引数に指定したファイルの内容を標準出力に書き出します。この場合、特に出力先の変更がないのでディスプレイに表示されます。sortコマンドは、標準出力から読み込んだ内容をソートして標準出力に書き出します。「<」と「>」を使えば入出力の対象をファイルにすることができます。標準出力先の変更に「>>」を使えば、追加モードになります。

■グループ化

複数のコマンドをグループ化して、各コマンドに対して共通の指定をしたり、環境の影響の範囲を制限することができます。

```
% (ps;who) > outfile ⏎
% cat outfile ⏎
PID  TT  STAT  TIME  COMMAND
765  p0  S     0:02  -csh(csh)        } psの出力
919  p0  R     0:00  ps
root      console Oct 29 10:24
funamoto  ttyp0   Oct 29 10:21(cs102) } whoの出力
susumu    ttyp1   Oct 29 10:29(vax01)
%
```

この例ではpsコマンドとwhoコマンドを連続実行して、この出力を1つのファイルに書き出しています。

■ワイルドカード

ファイル名を指定するときに、あるパターンを使えば複数ファイルに対して一括の指定をすることができます。このパターンのことを

ワイルドカード

と呼んでいます。ファイル名の指定を行うときに、その文字列の一部またはすべてを特殊文字を使って表現します。

特によく使われる

*

は、0個または任意の長さの文字列を意味する特殊文字です。例えば

*file

に一致するものとしては

file

exfile

1file

などがあるわけです。ただし、文字列の先頭がピリオド「.」ではじまるファイル名は除外されますから

.file

は対象とはなりません。

?

は、任意の1文字に一致します。

f?le

の指定をすれば

file

fale

f9le

f@le

などと一致することになります。

```
% ls ⏎
exfile1.c   exfile1.dat   exfile2.c   exfile2.dat   exfile33.dat
% ls *.c ⏎
exfile1.c   exfile2.c
% ls exfile?.dat ⏎
exfile1.dat   exfile2.dat
%
```

*.cというファイル名の指定は、拡張子がcであればその前に記述されている文字列がどのようなパターンでもよいわけです。

exfile?.datの「?」は、あくまで1文字だけのパターンに一致するだけですから、exfile33.datは対象になりません。

●ワイルドカード文字

*	0個以上の任意の文字列に一致
?	任意の1文字に一致
[*str*]	文字列(*str*)の内の1文字に一致
[*chr1*-*chr2*]	2つの文字の範囲の1文字に一致
{*str1*,*str2*,・・・}	文字列の内の1つに一致
~	ホームディレクトリに一致

Cシェルの便利な機能

■ヒストリ機能

過去に実行したコマンド行を保存しておいて、これを再使用したり参照することのできる機能を

ヒストリ

と呼びます。以前に入力したコマンドを単に再使用するだけでなく、一部を修正したり、文字列を追加してから実行させることもできます。長いコマンド列を入力したにもかかわらず、些細なミスをしたときには、この部分だけを修正するだけで簡単にコマンドの再入力ができます。

以前入力したコマンドを単純に表示するだけならば

history コマンド

を使います。

```
% history 5⏎ ←――― コマンド履歴を新しいものから5つ分表示する
22  find /usr/users/funamoto -name core -print
23  cat exfile
24  sort exfile-a | cat - exfile-b
25  cp exflie-b exfile-c
26  history 5
%
```

表示されるコマンド行のリストの先頭には、それぞれ番号が付けられています。その時点で最も値の大きなものが直前に入力したコマンド行です。

コマンド行を再利用する場合には、特殊文字

!

を入力コマンドの先頭に付けて使います。この直後にさまざまな指定をするのがヒストリの参照の基本的な書式です。

まず、最も使用頻度が高いのが直前のコマンドをそのまま再度実行する

!!

です。

```
% ls⏎
exfile1   exfile2   exfile3
% !!⏎
ls      ←――― 実行されるコマンドが表示される
exfile1   exfile2   exfile3
%
```

ヒストリリストの番号を指定すれば、任意のコマンド列を参照できます。例えば、23番目のコマンド列を実行するには

!23

のように入力します。

```
% !23 ⏎
cat exfile
Susumu Funamoto
Shingen Takeda
Masamune Date
%
```

コマンド列中にあるキーワードによって参照する方法もあります。例えば、コマンド列の先頭にcatがあるものを実行するには

!cat

と入力します。

```
% !cat ⏎
```

ヒストリリストの中に該当する行が複数あるときには、より最近のものが選択されます。この場合には、24番目のコマンドになります。

以上は以前のコマンド行をそのまま利用する代表的な例ですが、入力ミスをしたときなどをはじめとして、コマンド行中の一部を変更して利用したいこともあります。コマンド行を編集する機能を使うには

編集用サブコマンド

を使います。

編集の際の入力の書式は、基本的に

!*event* [:*position*] [:*action*]

のように「:」で区切られた3つの部分からできています。*event* は参照の指定でヒストリリスト中の特定のコマンド行を選択し、*position* でコマンド行中の編集対象の位置を指定します。*action* では、サブコマンドを使った編集内容を記述します。なお、*position* と *action* は省略することができます。

```
% ls exfile1 exfile2 /usr/funamoto/exprog.c ⏎
exfile1
exfile2
/usr/funamoto/exprog.c
```

```
% history 2 ⏎
      15 ls exfile1 exfile2 /usr/funamoto/exprog.c
      16 history 2
% !15:0 ⏎
ls      ←―――― 15番目のコマンド行のコマンド名部分だけが実行される
exfile1 exfile2 /usr/funamoto/exprog.c
%
```

この場合、*event* として15番目のコマンド行の実行（!15）を、*position* としてコマンド名（0）を指定していますから、結局15番目のコマンド行にあるコマンド名の部分だけを再実行します。

```
% cat !15:1-2 ⏎ ←―――― catコマンドの引数として、15番目のコマンド行の
cat exfile1 exfile2        引数の1番目、2番目を実行する
Susumu Funamoto    ┐
Shingen Takeda     ├ exfile1 の内容
Masamune Date      ┘
Shotaro Funamoto   ┐
Katsuyori Takeda   ├ exfile2 の内容
Tdamune Date       ┘
%
```

これは、catコマンドの引数に15番目のコマンドの引数の一部をもってくる例です、「1-2」は引数の1番目と2番目を表します。

```
% cd !15:3:h ⏎
cd /usr/funamoto
%
```

15番目のコマンドの3つ目（3）の引数（/usr/funamoto/exprog.c）のファイル名を除いたパス名部分をcdコマンドの引数としています。

直前のコマンドについては、一部修正して実行するためには手軽な入力形式があります。

^str1^str2

これを使えば、直前のコマンド行の中の文字列（*str1*）を別の文字列（*str2*）に置換して実行されます。

```
% tali exprog.c ⏎
```

```
% ^li^il ⏎
tail exprog.c
```

event、*position*、*action* の各指定方法については、コマンドリファレンスのhistoryコマンド（p.211）を参照してください。

■別名機能

コマンドを入力するときに、多くのオプションを指定しなければならないために入力行が長くなったり、なじみのないコマンド名の使用に困難を感じる場合に、別名を付けて入力を容易にするには

aliasコマンド

が使えます。

```
% alias del rm ⏎
% del file1 ⏎          ←――――― ファイルfile1を削除する
```

rmコマンドにdelという別名を付ければ、これ以降は引数などもまったく同様に記述すれば、delを使ってrmの機能を働かせることができます。次の例のように、長い記述をしなければならないコマンド列に短いキーワードを付ければ、入力が簡単になります。

```
% alias cfd cc file1.c file2.c file3.c ⏎
% cfd ⏎          ←――――― ファイルfile1～3.Cをコンパイルする
```

環境設定のための変数

シェルが動作しているときには、コマンドを実行させるために必要な環境を特徴づける特別な変数の値を参照しています。つまりこの値を必要に応じて設定すれば、シェルの動作を制御することができます。このような変数を

シェル変数

と呼んでいます。このシェル変数には、ユーザが必要に応じて確保し、値を設定したうえで利用する

ユーザ定義変数

と、シェルが動作しているときに、あらかじめ決められた特別な環境設定をす

るための

定義済み変数

があります。

ユーザ定義変数には、シェル上でコマンドを使う際に、よく使われる文字列（例えば、ディレクトリ名など）を値として設定しておけば、入力の手間を省くことができます。定義済みのシェル変数は、すでに決まった名称の変数名が特定の機能を意味するように設定されています。

例えば、保存されるヒストリリストの行数を設定するには

history

という定義済み変数を使います。このようなシェル変数に値を代入するときには

set コマンド

を使います。

```
% set history=100 ⏎
```

この例のような設定を行うと、Cシェルが保存できるヒストリリストの最大値が100行に設定されます。現在の設定は、このセッション間だけは有効です。

このほかに、代表的な定義済みシェル変数としては、プロンプトとして表示する文字列を設定するための

prompt

シェルへの引数が格納される

argv

コマンドの終了ステータスが格納される

status

などがあります。

```
% set prompt="whoami(¥!)" ⏎ ←―――― プロンプトを変更する
funamoto(1)%
```

whoamiコマンドは、現在使用している自分のユーザ名を出力し、これに合わせて「¥!」という記述がヒストリリストでの番号を表示します。つまり、promptというシェル変数に対する設定で、プロンプトがユーザ名＋ヒストリリスト番

号に変更されるわけです。

このシェル変数に設定された値を参照するときにはechoコマンドを使いますが、この際にはシェル変数の先頭に「$」を付けなければなりません。

```
% echo $history ⏎
% echo $argv ⏎
```

このように、historyやpromptのようにユーザが設定して意味の生じる変数と、argvやstatusのようにシェル自身が自動的に設定する変数があることには注意が必要です。

また、変数自身の存在が意味をもつものがあります。これを

トグル型シェル変数

といいます。

ログイン中に CTRL + D を入力すれば、ログアウトされますが、これを防ぐためには、

ignoreeof

というシェル変数を設定します。

```
% set ignoreeof ⏎
% ^D
%          } CTRL + D を入力してもログアウトされない
```

この場合、ignoreeofには値を設定しません。この変数自身の存在で効力が発揮されます。

設定されているすべてのシェル変数の設定状況を知るためには、setコマンドで引数を指定しません。

```
% set ⏎ ←──── 設定されているシェル変数を表示する
argv      ()
cdpath    (/usr/funamoto/sys /usr/sys /usr/spool)
cwd       /usr/funamoto
history   50
home      /usr/funamoto
inc       /usr/include
mail      /usr/spool/mail/funamoto
notify
```

```
path      (. /bin /usr/ucb /usr/funamoto/bin /usr/bin)
prompt    %
savehist  50
shell     /bin/csh
status    0
term      vt100
user      funamoto
%
```

シェル変数の値を無効にするときには

unset コマンド

を使います。

```
% unset history ⏎
% unset ignoreeof ⏎
```

シェル変数とは別に、環境を設定する際に使われる

環境変数

があります。環境変数はCシェルだけに限らず、一般のプロセスが動作しているときの環境情報が設定されています。もちろん、Cシェル自身も一つのプロセスとして動作しますから、これらの環境変数の値を参照します。シェル変数との大きな違いは、環境変数の場合はすべてのプロセスで共通的に、設定・参照ができる大域的な性格をもっている点です。一方、シェル変数の値はそのシェルだけで有効な、限定されたものです。なお、変数名にはシェル変数とは異なり、大文字が使われます。

代表的な環境変数には、ユーザのホームディレクトリが格納される

HOME

ログイン時のシェル名が格納される

SHELL

など10個ほどあります。

環境変数の値を参照するには

printenv コマンド

を使います。

```
% printenv ⏎                               ←――――――― すべての環境変数の値を表示する
TERM=vt100                                 ←――――――― 端末名
HOME=/usr/users/funamoto                   ←――――――― ホームディレクトリ
SHELL=/usr/bin/csh                         ←――――――― ログインシェルはCシェル
USER=funamoto                              ←――――――― ログイン名は funamoto
PATH=/bin:/usr/ucb:/usr/users/funamoto/bin:/usr/bin
LOGNAME=funamoto                           ←――――――― コマンドサーチパス
PWD=/usr/users/funamoto                    ←――――――― ワーキングディレクトリ
EDITOR=/usr/bin/vi
MAIL=/usr/spool/mail/funamoto
LINK-TIMEOUT=3
EXINIT=set ai aw ic sw=4 redraw wm=4|map g G|map v ~~~~←┐
%                                        エディタ(vi)の環境設定 ┘
```

環境変数に値をセットするには

　　setenv コマンド

を使います。環境変数の設定を無効にするには

　　unsetenv コマンド

を使います。

環境設定のためのファイル

Cシェルでは、各ユーザが自分の作業環境を自由に設定するために特別なファイルを用意しています。各ユーザのホームディレクトリにある

　　.cshrc

　　.login

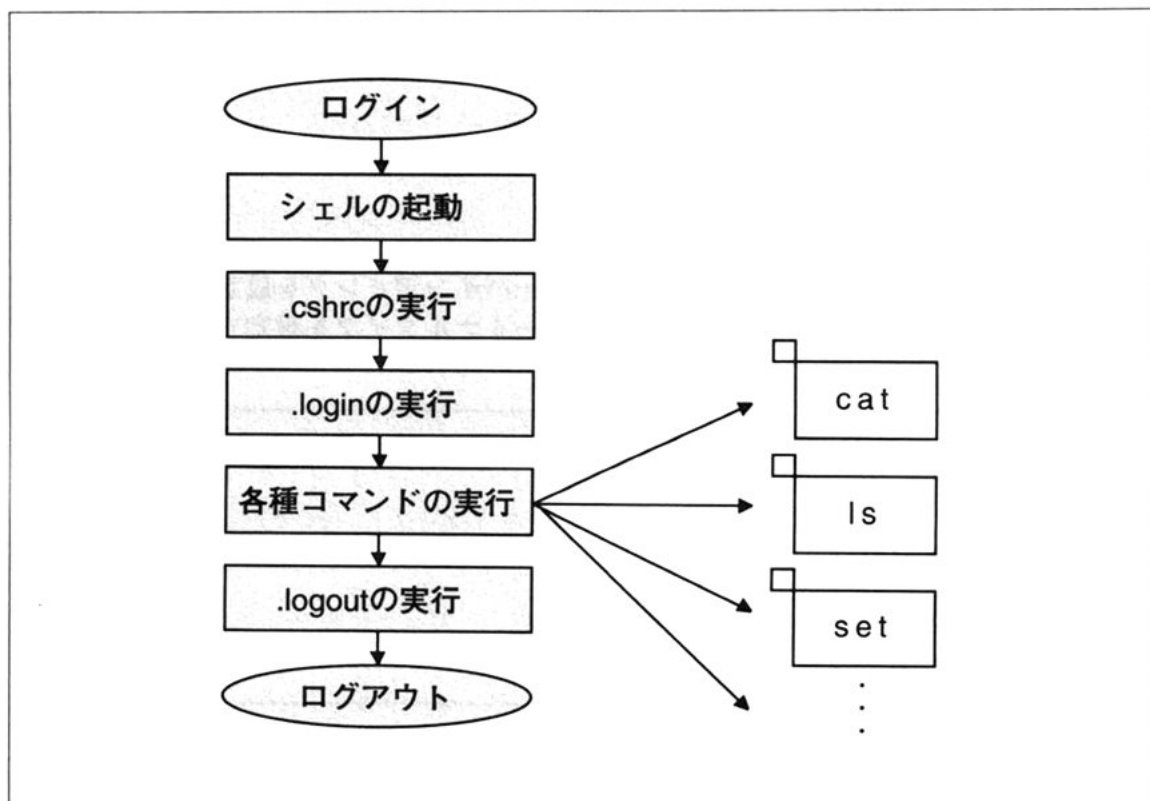
　　.logout

の3つのファイルには、環境設定をするためのコマンドが記述されています。これらは、オプションなしのlsコマンドではファイル名が表示されませんが、

　　-a オプション

を指定すれば表示されます。

```
% ls -a ⏎
.               .login          exfile          exprog10.c
..              .logout         exfile1
.cshrc          a.out           exprog.c
%
```

Cシェルが起動されると、まずこの.cshrcファイルが読み込まれて中に記述されたコマンドが実行されます。続いて、.loginファイルの内容が実行されます。また .logoutはログアウトの際に実行されます。

.cshrcファイルには、一般に

シェル変数・環境変数の設定（setコマンド、setenvコマンド）

エイリアスの設定（aliasコマンド）

プロンプトの設定（set prompt）

などが記述されます。これらの設定は、1回のセッションの間は有効ですから、毎回セッションの開始時に必要な設定を記述しておけば非常に便利です。

```
% cat .cshrc ⏎
set promt="`hostname`%"        ←―― プロンプトをホスト名＋％にする
set history=50                 ←―― ヒストリリストを50行にする
alias h history                ←―― ヒストリコマンドに別名を付ける
alias l 'ls -l'                ←―― ディレクトリコマンドに別名を付ける
%
```

セッションの開始および終了時に実行されるのが、それぞれ .login ファイルおよび .logout ファイルです。

.login ファイルには、主に

端末設定（stty コマンド）

が記述されます。

```
% cat .login ⏎
stty dec
stty erase ^H kill ^X intr ^C   ←―― キーバインディングを設定する
set term="vt100"                ←―― ターミナルタイプを設定する
%
```

.logout ファイルは、終了時の実行ですからそれほど意味のある設定はできませんが、

画面消去（clear コマンド）

終了メッセージ（echo コマンド）

などに使われます。

```
% cat .logout ⏎
echo "Good bye"
% ^D
Good bye

login:
```

.login と .cshrc の使い分けについては、前者は1回のログイン単位ですからこのセッションの間は設定を変える必要のないものを記述しますし、後者はシェル単位ですからシェルの起動の際に使用環境を変更したいときや変更しなければならないときに使います。

1・6

テキストエディタ vi と emacs

プログラムを開発したり、文書処理を行うときに必要なツールが、テキストを保存するファイルを作成したり、編集するための

　　エディタ

です。UNIX には、標準提供されているラインエディタ

　　ed,ex

とフルスクリーンエディタ

　　vi

があります。これらのエディタは、どんな UNIX システムでも利用できるようになっています。

最近では、テキストの入力や編集にはフルスクリーンエディタの利用が当たり前となっていますから、vi エディタの果たす役割は、非常に大きくなっています。この種のエディタとしては、MIT から PDS として配布されている

　　emacs

もプログラム開発を行うユーザの間ではかなり一般化してきています。emacs は、vi にはないマルチウィンドウをはじめとした、多くの機能をもっていますが、その分システムに与える負荷も高くなります。ですから、メモリ容量や CPU パフォーマンスに余裕のない環境では、emacs の利用は苦しいかもしれません。また、UNIX とは別に入手してインストールしなければなりませんから、どの UNIX 環境でも利用可能というわけにはいきません。

vi エディタ

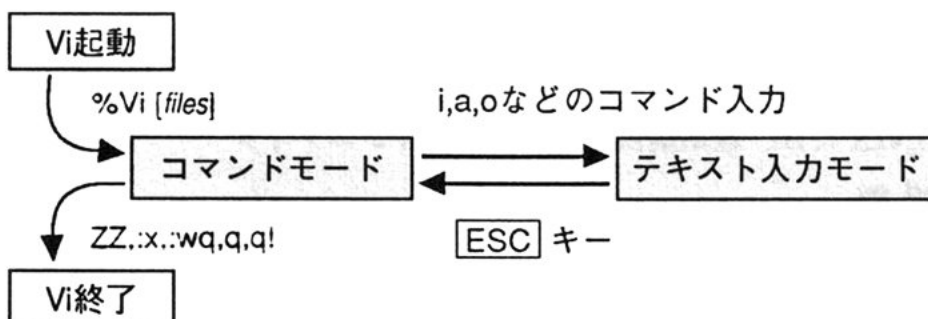
vi エディタは、ex エディタの機能をすべて備えた上位互換機能のあるフルスクリーンエディタです。

vi エディタには、テキストを入力する際の

テキスト入力モード

と、各種の編集をはじめとした機能を働かせるためのサブコマンドを入力する

　　　コマンドモード

があります。この2つのモードを適宜切り替えながら、入力および編集作業を行って、目的のテキストを完成させていきます。

　ここでは、以下に示す簡単なC言語のプログラムソースを新規に入力する例を使って説明していきます。

●例題プログラム

```
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
 {
        int c='¥000';

        while((getchar())!=EOF) {
                if (c!=DEL1 && c!=DEL2) {
                        putchar(c);
                }
        }
        exit(0);
}
```

　viエディタを起動をするには

```
% vi filename ⏎
```

と入力します。指定したファイル(*filename*)が存在しないときには、このファイル名で新規のファイルを作成することになります。

起動された直後のviは、

コマンドモード

になっています。このモードでは、だいたいキーボード上の1文字が、それぞれの機能と1:1対応になっていて、必要な処理を行わせることができます。

・カーソル移動
・データ入力指示(追加、挿入など)
・修正(消去、置換、複写、移動など)
・画面操作(画面移動、再表示、位置表示)

などがコマンドモードで使える機能です。

```
% vi testfile.c ⏎
```

●新規ファイルの起動画面

```
▮        ←―――― カーソルの位置
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"testfile.c"  [New file]
```

新規のファイルを指定したときの起動直後の画面には、最下行に新規ファイ

ルであることを示す

"testfile.c" [New File]

が表示されます。これ以外の行の先頭には、それぞれ「~」が表示されていて、テキストデータがないことを表しています。

また、エディタ上の現在の位置を示す

カーソル

が、1行目の1文字目にあります。このカーソルの位置が、テキストを入力する位置になります。

このviエディタを起動した直後は、まずコマンドモードになっています。このモードでは、いくらキーボードをたたいても画面には思いどおりの文字は表示されません。テキストを入力するにはコマンドモードからテキスト入力モードに切り替えなければなりません。

これには、通常はaまたはiのいずれかのviエディタのサブコマンドを入力します。それぞれ、現在のカーソル位置の直後、直前にテキストを挿入する際に使います。

ここでは、まだテキストが何もない状態ですから、aまたはiのいずれかを入力しても同じようにテキストを入力することができます。

一方、テキスト入力モードからコマンドモードに戻るためには、

[ESC]

を入力します。なお、今どちらのモードなのかについては、特別な表示があるわけではありません。もし、わからなくなってしまったときには、何回か[ESC]を入力すれば、必ずコマンドモードに戻れます。

さて、まず「例題プログラム」の1行目から入力するために、文字を挿入するための

aサブコマンド

を入力します。画面上は何も変化はありませんが、テキスト入力モードになっています。

```
#include <stdio.h>
```

を注意深く入力していきます。この行の最後の文字まで入力したら、[↵]を入力すれば次の行にカーソルが移動して、続けて次の行にテキストを入力するこ

とができます。

このまま間違いなく最後まで入力できれば、これで済んでしまいますが、実際には入力ミスをすることは少なくありません。そこで、編集機能を使ってミスを修正します。

例えば、1行目を

```
#includd <stdio.h>
```

のようにタイプミスをしたとします。「d」を「e」に変更しなければなりません。

一般に、編集の作業は、

1. 編集する位置にカーソルを移動する
2. 編集用のサブコマンドを入力する

の手順で行うのが基本です。

●カーソル移動

まず、カーソルを移動するためにはコマンドモードで、

h　　：左へ1文字移動
j　　：下へ1行移動
k　　：上へ1行移動
l　　：右へ1文字移動

のサブコマンドを適宜入力します。なお、h,j,k,l の各キーはキーボードの中央辺りに横一列にまとまって配置されています。

今この例では、1行目の入力を終わって ⏎ を入力後に気がついたとした場合、カーソルは2行目にあります。しかもテキスト入力モードですから、まず

ESC

を入力し、コマンドキーにしてから、

k

を入力し、カーソルを1行目に移動してから

l

を何回か入力すれば、修正する文字「d」の位置にカーソルがくるはずです。

●置換サブコマンド

ここで、タイプミスの「d」を「e」に置換します。置換をするサブコマンドには、

r　　：1文字を置換

R　　：文字列を置換

s　　：1文字を文字列で置換

S　　：行全体を置換

が使えます。ここではまず、

r

を入力後

e

を入力すれば「d」から「e」への置換が行われます。

●削除サブコマンド

この同じ修正をするために、誤った文字を削除した後、この位置に正しい文字を挿入するという方法もあります。削除を行うサブコマンドには

x　　：カーソルの位置の1文字を削除

X　　：カーソルの位置の左の1文字を削除

dd　　：カーソルのある行全体を削除

d$　　：カーソルのある行からこのファイルの最終行までを削除

などがあります。

前の例では削除する文字の位置に、カーソルを移動した後でサブコマンド

x

を入力して「d」を削除します。さらに、挿入のサブコマンドの

i

を入力した後に

e

を挿入して完成です。

さて、2行目以降の入力に戻ります。まず、

ESC

を入力した後で、次の行に挿入するためのサブコマンドの

o

を入力し、テキストを順に入力します。

●複写サブコマンド

ここで、複写の機能を使ってみます。2行目と3行目はよく似ていますから、

複写をして一部を修正することで、入力のタイプ量を減らすことができます。

まず2行目を入力した後に、この行にカーソルを置いたままで、

ESC

を入力してコマンドモードにします。ここで、複写する対象を指定するサブコマンドの

yy　：複写をする行を指定する

yw　：複写をする単語を指定する

を使い、さらに複写の実行をするサブコマンドの

p　：カーソルのある行の下にコピーする

pp　：カーソルのある行の上にコピーする

を続けて入力します。ここでは、

yyp

と入力すれば2行目と同じ内容が3行目に複写されます。

DEL1をDEL2に変更するために右へのカーソル移動の

l

を何回か入力して「1」の位置にカーソルをもっていき

r

を入力して文字を置換後、修正文字の

2

を入力します。

また、010を137に変更するにはさらに

l

を何回か入力し、最初の「0」の位置にカーソルをもっていって

r

に続けて

137

を入力します。ここで

ESC

を入力します。

●ファイル操作サブコマンド

このようにして一通り入力した後は、この内容をファイルに保存します。最

も簡単な方法は、コマンドモードで

ZZ（大文字に注意）

を入力することです。これでエディタも終了して、シェルのプロンプトが表示されます。このほかにファイル操作をするためのサブコマンドは、

:w[*file*]　：指定したファイル（*file*）に保存する

:wq　　　：ファイルに保存してエディタを終了する

:q　　　　：エディタを終了する

（ファイルに変更があると確認メッセージがでる）

:q!　　　：エディタを強制終了する（ファイルは保存しない）

:e[*file*]　：別のファイル（*file*）を編集する

などを使います。これらのサブコマンドは先頭にコロン「：」を付けて入力しますが、これはexエディタのコマンドを使うためのexモードにするために必要なのです。

ここでは、

:wq

を入力して、ファイルに内容を保存した後でエディタを終了します。

●文字列検索サブコマンド

このほかの代表的な機能として、文字列を検索するには、

/*str*　　　：順方向に文字列（*str*）を検索する

?*str*　　　：逆方向に文字列（*str*）を検索する

を使います。これらのコマンドを実行すると検索された文字列の先頭の文字にカーソルが移動します。

引き続き同じ文字列を検索するときには

n　　　　：順方向に再検索

N　　　　：逆方向に再検索

を入力します。

●画面操作サブコマンド

大きなファイルの中で編集する場所を探すためには、半ページや1ページ単位で表示画面を移動させることもできます。

^f　　　　：次のページを表示する

^b　　　　：前のページを表示する

^d　　　：半ページ次を表示する

^u　　　：半ページ前を表示する

個別のコマンドの詳細は、第4章で説明します。

emacsエディタ

emacsエディタは、viエディタに比べれば、

マルチウィンドウ

や

カスタマイズ

機能など、プロ仕様の機能をもつテキストエディタです。

emacsエディタは、標準ツールとしてUNIX上では必ずしも使えるようになっているわけではありませんが、インストールされていれば、

```
% emacs ⏎
```

と入力すれば起動することができます。

```
GNU Emacs 18.55.3 of Tue May 29 1990 on smra (berkeley-unix)
Copyright (C) 1988 Free Software Foundation, Inc.
Type C-h for help; C-x u to undo changes.  (`C-' means use CTRL key.)

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
You may give out copies of Emacs; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.
Type C-h t for a tutorial on using Emacs.

Nemacs version 3.3.1 of 1990.3.3
Type C-h T for a Japanese tutorial on using Nemacs.
For any other Nemacs specific information,
please read /usr/new/lib/nemacs3.3.1/etc/NEMACS.???.




 [----]-----NEmacs: *scratch*      (JJJ-:Lisp Interaction)--All----------
```

emacsの初期画面は、大きく3つの部分に分けられています。編集対象のテキストを格納しているバッファの内容を表示する**テキストウィンドウ**、表示されているバッファの名称やモードを表示する**モード行**、emacsからのメッセージや入力された文字列を表示する**エコー領域**があります。

通常編集されるテキストは、ファイル名と同じ名称のバッファに格納されていて、ここに対して編集が行われます。この編集内容をファイルに反映させるためには、そのためのコマンドを別に入力しなければなりません。バッファは複数使えますから、これに対応したテキストウィンドウを画面上に複数表示することもできます。

emacsのモードは、viエディタとは違ってキーとコマンドの対応関係や編集の様式によって設定されるもので、例えばC言語のソースリストの編集には、{と}の対応関係のチェックや適切なインデントなどを自動的に行ってくれる機能の種類のことをいいます。

起動時にファイル名を指定しない場合には、テキストウィンドウにオープニングメッセージが表示されますが、何かキー入力をすればクリアされて、モード行に名前が表示されているバッファscratchが表示されます。この時点では、バッファ中には何もテキストが存在しませんから、テキストウィンドウには何も表示されません。

emacsでは、入力されたすべての文字列がコマンドとして扱われます。この際に、一般のキーで入力された文字列は、バッファにこの文字列を挿入するコマンドとして機能しますから、ユーザにとっては入力文字がテキストウィンドウにそのまま表示されるように見えます。

一方、編集などを行うためのコマンドは、[CTRL]キーや[ESC]キー（または[META]キー）を付けた入力書式を使うのが一般的です。

viと同じ例題（p.68）で入力と編集の操作を説明します。

例題プログラムの1行目から入力するには、現在のカーソルの位置から

```
#include <stdio.h>
```

を直接入力していきます。この行の最後の文字まで入力したら、[⏎]を入力すれば次の行にカーソルが移動し、続けて次の行にテキストを入力することがで

きます。

なお、

```
#includd <stdio.h>
```

のようにタイプミスをした場合には、カーソルを移動するために、

CTRL ＋ b （左へ1文字移動）

CTRL ＋ n （下へ1行移動）

CTRL ＋ p （上へ1行移動）

CTRL ＋ f （右へ1文字移動）

などを適宜入力します。

ここでは、カーソルが2行目の先頭にいる状態からであれば、

CTRL ＋ p

を入力し、1行目に移動してから

CTRL ＋ f

を何回か入力すれば、修正する文字「d」の位置にカーソルがくるはずです。この文字を削除するには、

CTRL ＋ d （カーソルの位置の1文字を削除）

を入力します。さらに、変更する文字

e

を入力します。

2行目以降の入力に戻ります。まず、

⏎

を入力すれば、再び2行目の先頭にカーソルがきます。

ここで、複写の機能を使ってみます。2行目と3行目はよく似ていますから、複写をして一部を修正します。

まず2行目を入力した後に、

CTRL ＋ a

を入力して行の先頭にカーソルを移動します。ここで、

CTRL ＋ k

を入力すれば、このカーソル位置から行の末尾まで、つまりこの行全体が削除されます。この内容は、バッファに残っていますから、

CTRL + y

を2回入力して復活させれば、結果的にこの行を複写したことになります。あとは、先ほどと同様にカーソルを移動して変更を加えれば完成です。

このようにして一通り入力した後は、このバッファ内容をファイルに保存します。これには

CTRL + x CTRL + s

を入力します。エコー領域に

File to save in:~/

と表示されますから、これに続けてファイル名を入力して ⏎ を入力すれば、このファイル名で保存されます。

なお、すでに存在するファイルをemacsに読み込むには、

CTRL + x CTRL + v

または

CTRL + x CTRL + f

を使います。

emacsを終了するには、

CTRL + x CTRL + c

を入力します。バッファの内容が変更されたあとで、保存されていないときには警告メッセージが表示されます。

このほかの代表的な機能として、文字列を検索するには、

CTRL + s *str* ：順方向に文字列（*str*）を検索する

CTRL + r *str* ：逆方向に文字列（*str*）を検索する

を使います。これらのコマンドを実行すると、検索された文字列の先頭の文字にカーソルが移動します。

引き続き同じ文字列を検索するときには

CTRL + s ：順方向に再検索

CTRL + r ：逆方向に再検索

を入力します。

大きなファイルの中で編集する場所を探すためには、半ページや1ページ単位で表示画面を移動させることもできます。

CTRL + v　：次のページを表示する

ESC + v　：前のページを表示する

emacsの特徴のマルチウィンドウを操作するには、

CTRL + X 2　：ウィンドウを上下に分割

CTRL + X 0　：ウィンドウを消去

をはじめとして、各種のコマンドが用意されています。

そのほかには

CTRL + g　：コマンドの中止（ミス操作からの復帰）

CTRL + _　：1つ前の操作の取り消し

ESC + !　：シェルコマンド（*cmd*）の実行

などが編集中に役に立つコマンドです。

1・7

コミュニケーションと情報入手

電子メールを送る(mail)

同じシステムを利用しているユーザとメッセージを交換するには

mail コマンド

を使います。いわば、郵便の機能を果たしてくれるコマンドです。メッセージを送られたユーザは、ログインしたときに、メッセージが着信していることを示す表示によって気づきますから、間違いなくメッセージを読むことができます。

```
% mail funamoto ⏎        ←―― ユーザ funamoto にメールを送付する
Subject : apointment     ←―― メールの表題を入力する
On Friday I will visit you.                                  } メール内容の入力
The time will be around two o'clock in the afternoon. ⏎     }
^D   ←―――――――――――――― 入力の終了
%
```

標準入力（キーボード）からメッセージを入力します。なお、リダイレクションを使えば、ファイルからメッセージとして送ることもできます。

```
login : funamoto ⏎
Password:______ ⏎
Last login: Fri Sep 6 10:11:11 from sun01
SunOS Release 4.1.1-JLE1.1.1 (GENERIC) #1: Mon May 21 13:46:13 JST 1990
You have mail.   ←―― メール着信のメッセージ
%
```

ログインしたときに、プロンプトが表示される直前にメールの着信を通知するメッセージが表示されます。

メールの着信を確認したりメールを読むときは、引数の指定なしでmailコマンドを入力します。

```
% mail ⏎
No mail for funamoto ←—— メールが着信していないときのメッセージ
%
% mail ⏎
Mail version SMI 4.0 Fri Sep 20 10:44:02 JST 1990 Type ? for help.
"/usr/spool/mail/funamoto": 3 messages 3 unread
>U 1 ueno            Tur Sep 19 08:42   13/314  apointment
 U 2 date            Tur Sep 19 11:16   16/374  Let's have a party
 U 3 yamada          Fri Sep 20 09:57   16/399  introduce
& 2 ⏎
Message  2:                                到着しているメールの一覧
From date Tur Sep 19 11:16:37 1990
Return-Path: <date>
Received: by sun03. (4.0/SMI-4.0)
        id AA00406; Fri, 19 Sep 90 11:16 34 JST
Date: Tur, 19 Sep 90 11:16 34 JST
From: hash (M.date)
Message-ID: <9007060046.AA00406@sun03.>
To: funamoto
Subject: apointment
Status: R
On Friday I will visit you.
The time will be around two o'clock in the afternoon.  ←—— メール内容
& q ⏎ ←—— メールコマンドを終了する
%
```

複数のメールが届いている場合には、特に指定がなければ、古いものから順に表示されます。メールの番号を指定すれば、そのメールを選択して読むことができます。

mailコマンドには、いくつかのサブコマンドがあります。読み終わったメールで必要のなくなったものは、

dサブコマンド

を使って削除します。また、特に残しておきたいメールは、

sサブコマンド

を使ってファイル名を指定すれば、ファイルとして保存することができます。

　　q サブコマンド

は、mail コマンドを終了するときに使います。

　サブコマンドの一覧は、

　　[HELP] または [?]

を入力すると表示されます。

他のユーザとメッセージを交換する(write, ,talk mesg)

　同一の UNIX システムを利用しているユーザは、互いに他のユーザとの間でリアルタイムで情報を交換するための通信を行うことができます。

　ログイン中の他のユーザにメッセージを送るのが

　　write コマンド

です。

```
% write funamoto ⏎  ←――― ユーザ funamoto にメッセージを送る
Hello, Mr. funamoto.                                  } メッセージを入力
Could you meet me at your convenience this week?      }
^D   ←――― 入力を終了する
%
```

　このように、相手方に送るメッセージはキーボードから入力し、終了するときには、

　　[CTRL] + [D]

を入力します。

　これで、相手方の使っている端末のディスプレイに表示されます。送り元のユーザ名が表示され、続けて送信されたメッセージが順に表示されます。

```
%
Message from takeda@nws01 on ttyp1 at 18:33 ...
Hello, Mr. funamoto.
Could you meet me at your convenience this week?
EOF
%
```

ユーザがお互いに write コマンドを実行すれば、端末を通して会話することもできます。まず、一方のユーザが write コマンドを実行してメッセージを送ります。この際に CTRL ＋ D を入力せずに、そのままの状態で待ちます。他方のユーザは、このメッセージを受けた直後に、write コマンドを実行します。これで、双方向の通信ができる状態になります。どちらの側からも、改行を入力した時点でメッセージが送られますから、お互いにタイミングを取って送信すればよいわけです。この交信は、CTRL ＋ D が入力されるまで続けることができます。

write コマンドと同様に、2人のユーザがメッセージを交換するには、

　　talk コマンド

を使う方法もあります。使い方は、2章のリファレンスを参照してください。

write コマンドは、送り先のユーザがどのような作業をしていても、強引にメッセージを送ってきます。これは、受信側としてはつごうの悪いときもあります。これに対処するために使われるのが、メッセージの受け取りの拒否や許可を指定するための

　　mesg コマンド

です。

オプションには、メッセージの受け取りを許可をするための

　　y

または、拒否をするための

　　n

を指定します。

```
% mesg ↵          ←――――― 現在の状態を表示する
is y              ←――――― メッセージ着信許可
% mesg n ↵        ←――――― メッセージ着信拒否に設定する
% mesg ↵
is n
%
```

他のユーザの使用状況（who）

ログインしている他のユーザと write コマンドなどを使って会話をするには、そのユーザが現在システムにログインしていなければならず、まずそのことを確認しなければなりません。現在ログインしているユーザのログイン名を表示するためには

who コマンド

を使います。

```
% who ⏎
funamoto     ttyp0   Sep 19 11:21
yamada       ttyp1   Sep 19 11:22   (cs124)   3ユーザがログインしている
ueno         ttyp3   Sep 19 11:54   (cs123)
%
```

現在ログインしているユーザのログイン名、端末名、ログイン日時を画面に表示します。

who コマンドにオプション

am i

を指定すると、自分に関する情報だけが出力されます。

```
% who am i ⏎
sun03!funamoto   ttyp0   Sep 19 11:21
%
```

1・8

ネットワーク環境

UNIX には、その誕生の当時から基本的なネットワーク機能が提供されていました。

UUCP（unix to unix copy）

という、

RS-232C

のシリアル回線を利用して、UNIX 同士の通信を行う機能です。

コマンドとしては、ローカルシステムとリモートシステムの間でファイルを転送する

uucp コマンド

とリモートシステム上で任意のコマンドを実行させる

uux コマンド

があります。

これらの機能は RS-232C ケーブルによる接続ですから、安価で手軽に実現できますが、転送スピードや信頼性の面では取り扱いにくい点もあります。

最近では、特にワークステーションの場合には

Ethernet（イーサネット）

と呼ばれるネットワークをベースとした

LAN

を介在させ、一人1台を基本としてお互いの資産を利用する形態が一般化しています。この環境を利用するには、強力な支援機能が必要になります。

UNIX は、ユーザがこのネットワークを容易に利用することができるように、

リモート端末

リモートファイル転送

リモートジョブ実行

という機能を提供しています。「リモート」は、現在自分が使っているマシンを

「ローカル」としたときに、このマシン以外で同じネットワークに接続されているコンピュータを意味させるための用語です。

自分のマシン（端末）を他のマシンの端末として使うためのリモート端末機能にはrloginコマンド、他のマシンに自分のマシンからファイルを転送するにはrcpコマンド、他のマシン上で自分のマシンからジョブの実行依頼をするにはrshコマンドを使います。

これらの機能を利用するには、BSD系のUNIXが必要ですが、最近ではSystemV系でも使えるようになっています。

●ネットワーク機能

	RS-232C	Ethernet
ファイル転送	uucpコマンド	rcpコマンド
リモート端末	tipコマンド	rloginコマンド
リモートジョブ実行	uuxコマンド	rshコマンド
ファイルシステム	――	NFS
ウィンドウシステム	――	X Window System
その他情報収集など	――	rwho,ruptimeなど

リモート端末（rlogin）

自分の使っているローカルマシンから別のリモートマシンにログインする機能を

リモート端末

といいます。つまり、ローカルマシンに接続された端末をリモートマシンの端末として使うことができるのです。リモートマシンにある資源や環境は、ローカルマシンとして直接ログインする場合と同様に有効です。すなわち、わざわざ離れたリモートマシンの場所までユーザ自身が移動しなくても、手近にあるマシンからリモートの環境を利用できます。

この機能を使うためには、ローカルマシンにログインした状態で、

rloginコマンド

を使います。

ホスト名がlcl01のローカルマシンから、rmt01のリモートマシンにログイン

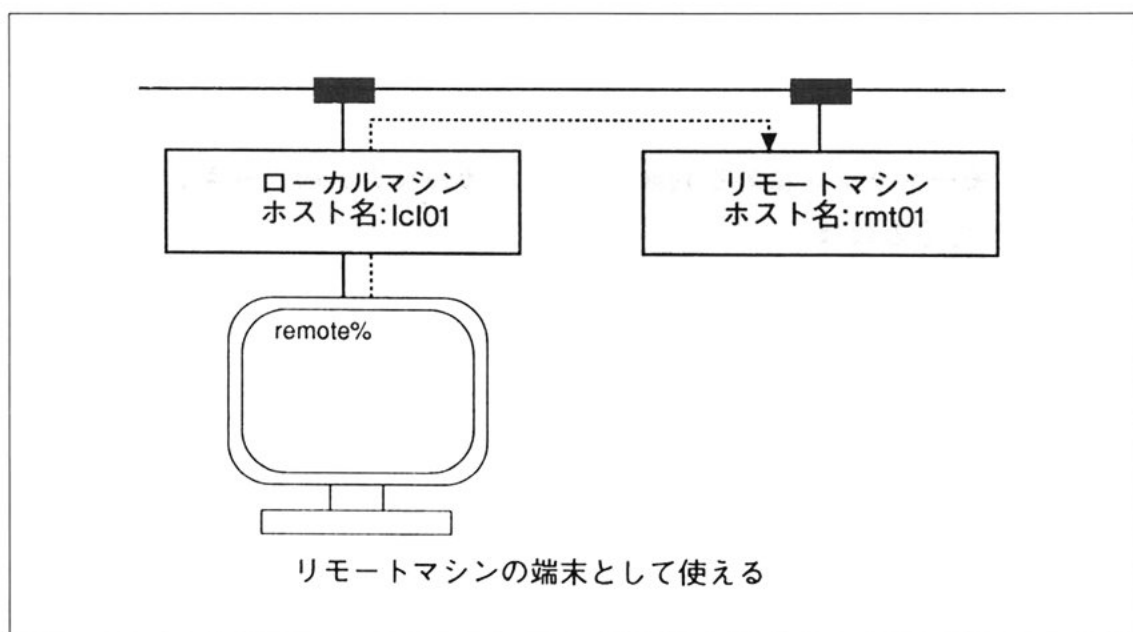
するには

```
lcl01% rlogin rmt01 ⏎
Last login: Tue Nov 26 18:33:26 from lcl01
NEWS-OS Release 4.0R #0: Tue Dec 18 19:02:18 JST 1990
rmt01%
```

のようになります。これ以降は、rmt01 上に直接ログインしたのと同じように、rmt01 の環境を使用できます。

システムによっては他のマシンからのログインを制限する設定をしている場合もありますが、このような設定についてはシステム管理者(スーパーユーザ)に問い合わせてください。

●リモート端末

リモートマシンの端末として使える

リモートファイル転送(rcp)

ネットワークに接続されているマシンを使って作業をするときには、それぞれのマシン間で相互の資産を共有することが必要不可欠です。ローカルマシンのディスクにあるファイルをリモートのマシンに転送したり、またその逆にリモートマシンにあるファイルをローカルマシンに持ってくるときには、

リモートファイル転送

の機能を使います。

これを利用するには、

rcp コマンド

を使います。

基本的な使い方は、cp コマンドと同様ですが、ファイル名(file1 または file2)の指定には、リモートマシンのホスト名を付けて

hostname : filename

の書式を使います。

ローカルマシン lcl01 のワーキングディレクトリにあるファイル exfile1 を、リモートマシン rmt01 のディレクトリ/tmp に転送するには、

```
lcl01% rcp exfile1 rmt01:/tmp ⏎
```

となります。

逆に、リモートマシンにある testfile をローカルマシンのワーキングディレクトリに持ってくるには

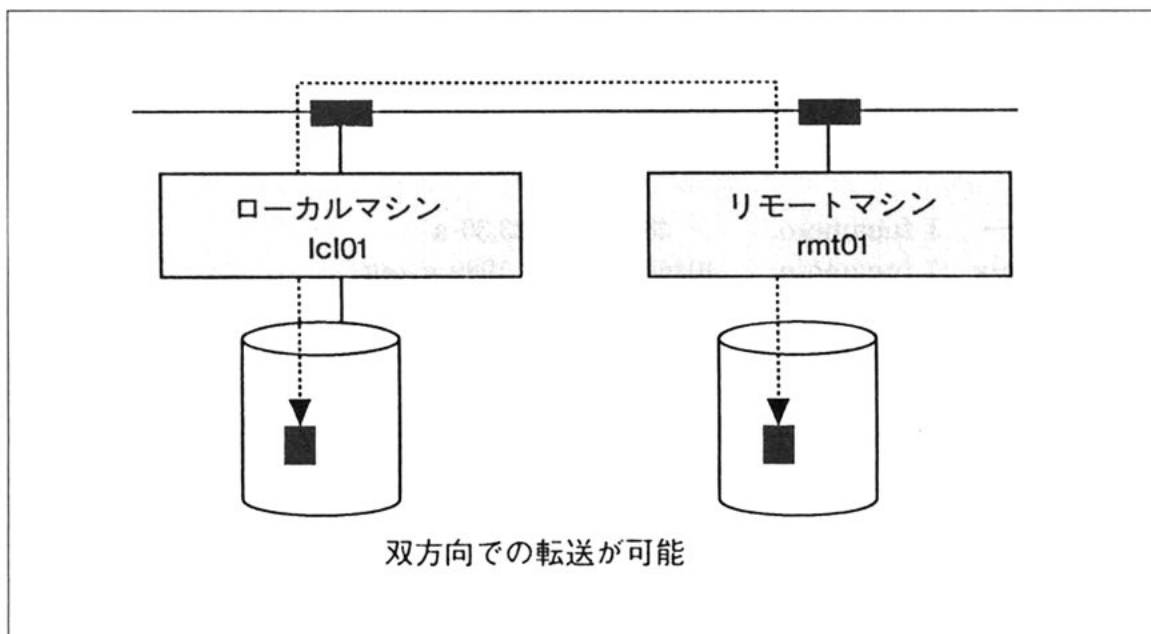
```
lcl01% rcp rmt01:testfile. ⏎
```

となります。ここで、testfile は rmt01 上のホームディレクトリにあるファイルです。

さらには、ローカルマシン上でリーモートマシン同士のファイル転送をすることもできます。

```
lcl01% rcp rmt01:testfile1 rmt02:testfile2 ⏎
```

これは、lcl01 上で rmt01 の testfile1 を、rmt02 の testfile2 にコピーする例です。双方のファイルはそれぞれのマシンのホームディレクトリにあります。

●ファイル転送

リモートシェル(rsh)

リモートマシンで自分のジョブを実行させたいときには、リモート端末機能を使います。ただし、せいぜい1個程度のコマンドを実行させたいときには

リモートシェル（リモートジョブ実行）

のほうが使いやすいのです。逆に、多くのコマンドを実行するときに、リモートシェル機能を使えば、毎回のようにログインの手続きが実行されますから、効率が悪くなります。特別な場合を除いてはまず必要ありません。

リモートシェル機能を使うには、

rsh コマンド

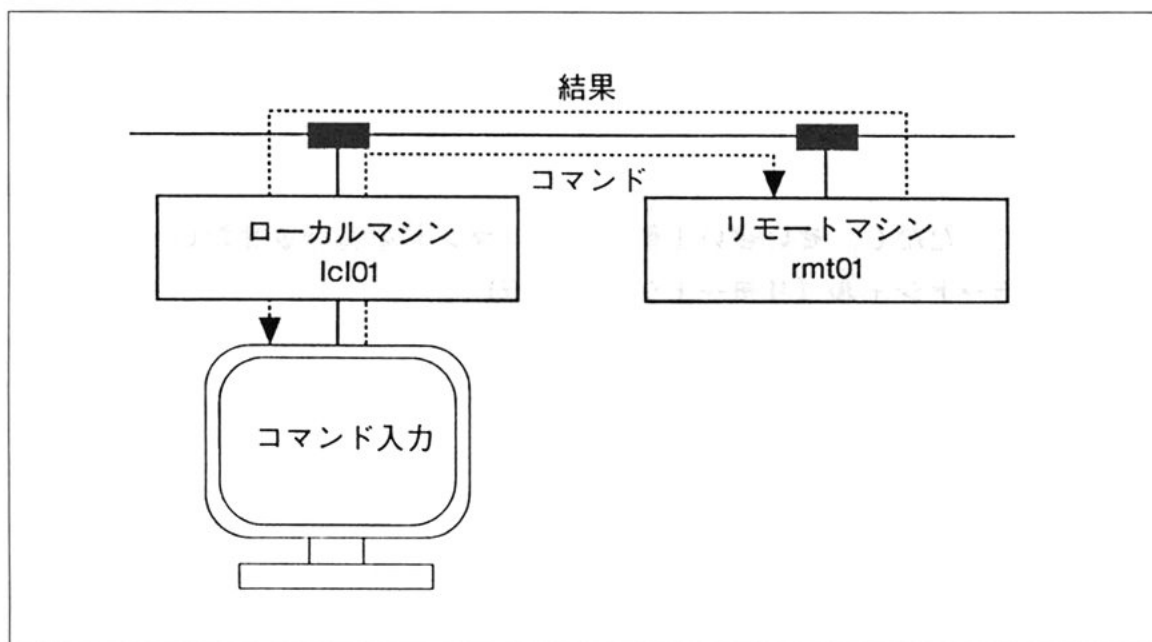
を使います。

また、リモートマシンで実行されたジョブの結果をローカルマシンに欲しい場合があります。ところが、リモート端末機能を使った場合には、リモートマシン上ですべてが実行されますから、ローカルマシンで結果を直接受け取ることはできません。リモートシェルでは、パイプとリダイレクションを使ったときには、これを指定する「|」や「>」や「<」の前まではリモートマシンで処理されますが、これ以降のコマンドはローカルマシンで処理されますから、

リモートマシンで実行された結果をローカルマシンで実行されるコマンドに受け渡すこともできます。

```
lcl01% rsh rmt01 ls -l | sort > exfile ⏎
lcl01% cat exfile ⏎
total 42
-rw-r--r--   1 funamoto        28 Oct 16 23:30 a
-rwxr-xr-x   1 funamoto     40464 Oct 30  1989 a.out
-rw-r--r--   1 funamoto       174 Oct 16 23:30 exfile.c
-rw-r--r--   1 funamoto       273 Oct 28 23:27 exfile.dat
lcl01%
```

となります。ls -l はリモートマシン rmt01 で実行され、この結果はパイプを通してローカルマシン lcl01 で実行される sort コマンドに受け渡されます。当然 exfile は、ローカルマシンのワーキングディレクトリに作られます。

●リモートジョブ

ネットワーク情報の入手(rwho,ruptime)

ネットワークに接続されたさまざまなマシンを利用するユーザにとっては、それぞれのマシンや同じネットワーク内で作業をしているユーザの状況を把握

することも必要になってきます。

ネットワーク内のユーザの使用状況を知るためには

rwho コマンド

を使います。これは、who コマンドのネットワーク版です。

表示される情報は、ユーザ名、端末名、ログイン時刻ですが、端末名にはホスト名が付けられています。

● rwho コマンドの例

```
% rwho ⏎ ←────────── ネットワーク上のユーザ情報を表示する
user0012  sun20:console   Dec 11 15:10:24
user0012  sun20:ttyp0     Dec 11 15:10:23
user0012  sun20:ttyp1     Dec 11 15:11
user0033  sun12:console   Dec 11 15:13
user0042  sun16:console   Dec 11 15:06:18
user0042  sun16:ttyp0     Dec 11 15:17:17
user0042  sun16:ttyp1     Dec 11 15:17:17
user0042  sun16:ttyp2     Dec 11 15:17
user0042  vax03:ttyp0     Dec 11 13:31:29
   :          :              :
user1014  vax03:tty00     Dec 11 13:10:01
user1017  vax03:tty03     Dec 11 13:13
user1018  vax03:tty04     Dec 11 13:14
user1026  vax02:tty02     Dec 11 13:12
user1027  vax02:tty00     Dec 11 13:12
user1028  vax02:tty04     Dec 11 13:16:04
user1032  vax02:tty01     Dec 11 13:13:03
user1033  vax02:tty03     Dec 11 13:12
user1035  vax01:tty11     Dec 11 13:08:02
user1057  vax01:tty13     Dec 11 13:16:05
user1058  vax01:tty12     Dec 11 13:15
user1059  vax04:tty04     Dec 11 13:17
user1060  vax01:tty08     Dec 11 13:08
user1075  vax01:tty09     Dec 11 13:05:01
%
   ↑         ↑    ↑             ↑
ユーザ名  ホスト名：端末名    ログイン時刻
```

一方、ネットワーク上のマシンの CPU に関する情報を入手するためには

ruptime コマンド

を使います。これは、ネットワーク対応の uptime コマンドです(2 章コマンド・リファレンス参照)。

マシンにかかっている負荷状況やユーザについての情報を出力することができます。

● ruptime コマンドの例

```
% ruptime ⏎ ←──── ネットワーク上のシステムのロードアベレージを表示する
sun01        up      7:14,   0 users,   load 0.01, 0.00, 0.00
sun02        up      7:14,   0 users,   load 0.01, 0.00, 0.00
sun03        up      7:14,   1 user,    load 0.34, 0.75, 0.57
sun04      down   2+21:24
sun05      down   2+21:25
sun06      down   2+21:26
sun07        up      7:16,   1 user,    load 1.28, 1.28, 0.80
  :           :         :       :          :     :     :     :
sun17        up      2:05,   3 users,   load 0.00, 0.03, 0.00
sun18      down   4+20:38
sun19      down   4+20:16
sun20        up      0:26,   3 users,   load 0.16, 0.20, 0.00
vax01        up      7:12,   7 users,   load 1.85, 2.16, 2.16
vax02        up      7:13,   5 users,   load 0.84, 1.20, 1.27
vax03        up      7:13,   6 users,   load 0.42, 0.69, 0.74
vax04        up      7:12,   7 users,   load 2.64, 2.74, 2.64
vs201        up      7:16,    users,    load 0.23, 0.37, 0.25
%
 ↑            ↑        ↑        ↑                 ↑     ↑     ↑
ホスト名     動作中    時間     ユーザ数          1分   5分   15分
            ／停止中                             ロードアベレージ
```

ネットワークファイルシステム(NFS)とイエローページ

ネットワークファイルシステムは、ネットワーク上にあるリモートのマシンのディスク（リモートディスク）を、ローカルのマシンのディスクと同一にローカルディスクとして使えるようにする機能をいいます。

最も一般的に使われているのが、サンマイクロシステムズの

NFS

というシステムです。ユーザからは何も意識せずに、自分が使っているマシンのディスクとして使えるわけですから非常に便利です。

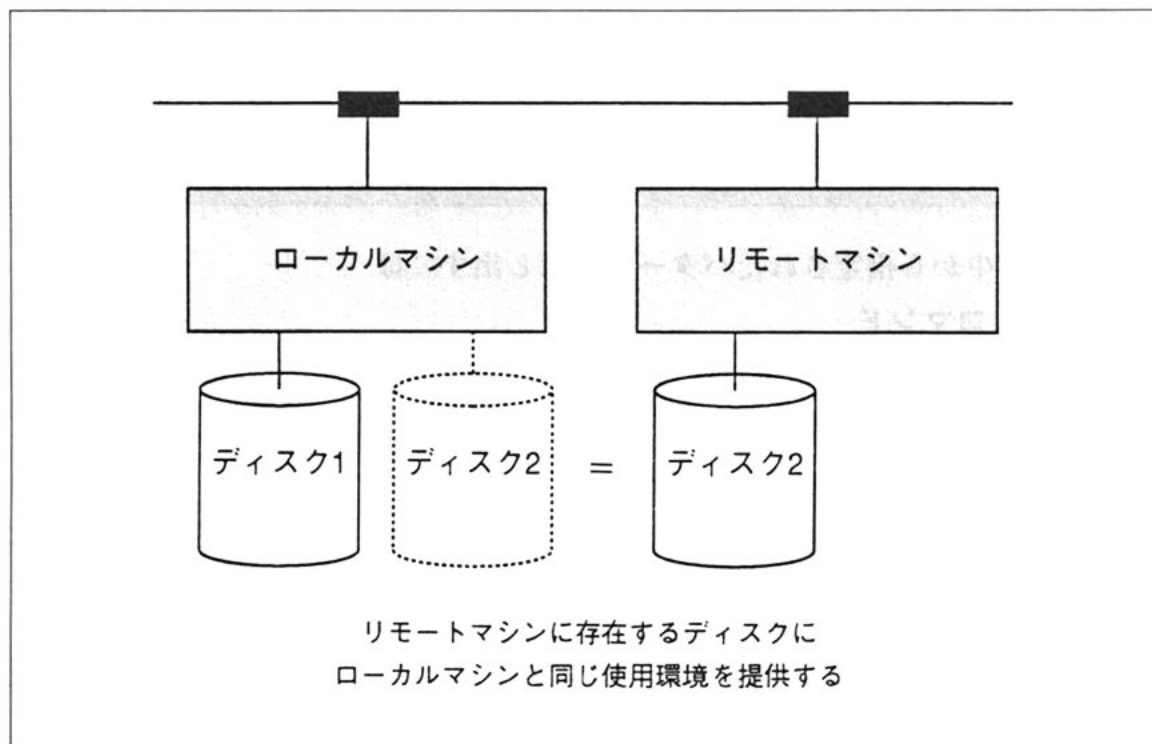
この機能を使うことによって

・データの一貫性保持

・特定のマシンからの束縛解消

・ディスクスペースの節約

などの利点を得られます。

● NFS の概念図

NFS は、主としてユーザファイルとアプリケーションソフトの共有化に威力を発揮します。一方、システムの管理情報、例えばパスワードファイルの一貫性の保持や保守の効率化をするために

イエローページ（YP）

という機能があります。YP では、ネットワーク上に YP サーバと呼ばれるネットワーク全体の管理情報をもつマシンを設定し、一手に管理を引き受ける役割をもたせます。これも、ユーザにとってはまったく意識せずに使える機能です。

1・9

パターン処理

UNIX のコマンドのなかには、ファイル中のデータに対してさまざまな編集操作をする機能をもつものが数多くあります。このなかでも、特にユーザがファイル中に存在するデータのパターンを指定することによって、特定の文字列に対して多彩な編集を行う機能を提供するコマンドがあります。

パターンの検索(grep)

ファイル中から指定されたパターンを探し出すには

grep コマンド

を使います。パターンの指定には、grep コマンドの名称の由来の global regular experssion printer でいう、

正規表現（regular experssion)

が使われます。正規表現の最も単純な使い方は、パターンとして探したい文字列をそのまま指定するものです。

```
% cat exfile.dat ⏎
Funamoto Susumu     Tokyo       (3722)1234
Date Masamune       Miyagi      (77)4567
Takeda Shingen      Yamanashi   (34)8901
Funamoto Syotaro    Tokyo       (3333)4321
Date Tadamune       Sendai      (79)1357
Takeda Katsuyori    Suwa        (23)2468
%
```

例題として使うファイル exfile.dat には、名前、都道府県名、電話番号を記述した個人データが 6 件分格納されています。

```
% grep Tokyo exfile.dat ⏎ ←———— パターン Tokyo を含む行を表示する
Funamoto Susumu      Tokyo        (3722)1234
Funamoto Syotaro     Tokyo        (3333)4321
%
```

パターンの文字列の指定で問題になりそうなのが、文字列の中にスペースが含まれる場合です。シェルは、スペースをコマンド列の中で区切りをするための特殊文字として解釈してしまいます。こういう場合には、

```
% grep 'Funamoto Susumu' exfile.dat ⏎
Funamoto Susumu      Tokyo        (3722)1234
%
```

のように検索文字列をシングルクウォート「'」で囲みます。ダブルクウォート「"」でも構いません。これで、シェルはスペースを通常の文字として認識します。

正規表現では、検索するパターンを正確に覚えていなかったり、複数のパターンをまとめて検索するときには、

メタキャラクタ

を使うことができます。

Funamoto と funamoto のように大文字と小文字の区別をしないで、双方とも検索対象とするためには

```
% grep "[fF]unamoto" exfile1.dat ⏎
Funamoto Susumu 36 Tokyo
susumu funamoto Nippon Electronics College
%
```

のようにメタキャラクタのブラケットを使います。この中の文字のいずれか1文字と照合するという機能を果たします。

grep コマンドに改良を加えたものとして、パターン指定を単純化することによって処理の高速化を図った

fgrep コマンド

とパターン指定を拡張した

egrep コマンド

があります。これらの使用例については、コマンドリファレンスのgrepコマンド（p.204）を参照してください。

パターン走査と処理(awk)

grepコマンドがファイル中にある特定のパターンを含む行を抽出するだけなのに対して、抽出された行に対して各種の編集を施せる機能をもつのが、

awkコマンド

です。

awkコマンドでは、C言語のプログラムに似た構文でプログラムを記述し、制御構造を解釈できますから、通常のプログラム言語にかなり近い処理手続きを実行させることができます。もちろん変数が使えますし、数値の計算や演算子も使えます。

まず、grepコマンドと同様のファイル中のパターンを検索してみます。

```
% awk '/Tokyo/  {print $1,$2,$3,$4} '  exfile3 ↵
Funamoto Susumu    Tokyo     (3722)1234
Funamoto Syotaro   Tokyo     (3333)4321
%
```

前節のgrepと同じ出力結果が得られます。しかし、grepのコマンド行に比べれば難しい記述になっています。これは、出力についてはawkコマンドのほうがさまざまな指定ができるようになっているために、単純な処理についてはかえって冗長な記述になってしまうことがあるからです。

awkのプログラムの一般化された書式は、次のとおりです。

*ptn*1 {*action*1}

*ptn*2 {*action*2}

⋮

ptnn {*actionn*}

指定したパターン（*ptnn*）に一致した行に対して、それに続いて{ }で囲まれて記述されているアクション（*actionn*）を実行します。このパターンの指定（*ptnn*）は、スラッシュ「/」で囲みます。

このコマンド行のアポストロフィ「'」で囲まれたプログラム部分では、一致したパターンを出力する命令

print

を使っています。$n は、一致した行の各項目（フィールド）を値としてもつ、左からの順番で番号を付けた変数です。この変数のことを

シンボル

といいます。

```
Funamoto   Susumu   Tokyo   (3722)1234
   $1        $2       $3        $4
                 $0
```

$n というシンボルは、n 番目のフィールドを意味しますが、

$0

はその行全体を表します。このほかにも、シンボルには

NF　　　　：フィールドの数
NR　　　　：入力行の数
FS　　　　：入力フィールド区切り文字
RS　　　　：入力行の区切り文字
OFS　　　：出力フィールドの区切り文字
ORS　　　：出力行の区切り文字
FILENAME：入力ファイル名

などがあります。区切り文字というのは、フィールドとフィールドの間の区別をつけるための文字のことです。通常は、スペースやタブが使われます。

出力をする順番は、容易に変更することができます。ここが、grep コマンドとは違うところです。print 文に続いて出力するシンボルの順番を変えます。

```
% awk '/Tokyo/  {print $4,$2,$1} ' exfile3⏎
(3722)1234   Susumu   Funamoto
(3333)4321   Syotaro   Funamoto
%
```

この例では、4 番目のフィールドの電話番号を先頭にして、続いて 2 番目の名前と 1 番目の名字の順に表示させ、3 番目の住所の出力は省略しています。

// で指定するパターンを省略したときには、入力したファイルのすべての行が操作の対象になります。ですから、単に項目の並べ替えをするだけにも使うことができます。

```
% awk ' {print $3,$4,$1,$2} ' exfile3 ⏎
Tokyo     (3722)1234     Funamoto Susumu
Sendai    (77)4567       Date Masamune
Koufu     (34)8901       Takeda Shingen
Tokyo     (3333)4321     Funamoto Syotaro
Sendai    (79)1357       Date Tadamune
Suwa      (23)2468       Takeda Katsuyori
%
```

awk コマンドのプログラム中では、制御文、組み込み関数、演算子が使えます。これらの詳細については、コマンドリファレンスの awk コマンド(p.122)を参照してください。

awk で使えるプログラムに柔軟性を与えるための構造として、実行の際に一度だけ行われる

BEGIN

とすべての処理が終わったあとの後処理をするための

END

という特殊なパターンを使うことができます。

これを使ったときのプログラムの書式は、

```
BEGIN {
            ACTION
      }
      {
            ACTION
      }
END   {
            ACTION
      }
```

となります。

例えば、

```
if (($1 < 1)&&($1 <3))
   {
     print $2
   }
else
   {
    if ($1 >=3)
       {
        print $3
       }
   }
}
```
（if 〜 } までがプログラムの部分）

```
% cat infile⏎
Susumu    m    100    90     80
Kenichi   m    90     100    80
Masako    f    80     100    90
tarou     m    90     80     100
Reiko     f    100    100    100
%
```

こうなってくると本格的にプログラムらしくなってきますが、awkのコマンド行の中に記述するには多すぎます。そこで、プログラム部分をファイルにしておいてawkに読み込ませてこれを実行するために

-f オプション

を使います。

awk -f *progfile file*

引数には、プログラムファイル（*progfile*）と処理対象ファイル（*file*）を指定します。

この書式を使って、比較的大きな処理を行うawkのプログラムを見てみます。いま次に示すような成績ファイルがあるとしてawkで成績処理を行ってみます。

●成績ファイル infile

```
% cat infile⏎
Susumu    m    100    90     80
Kenichi   m    90     100    80
```

```
Masako   f    80    100    90
tarou    m    90    80     100
Reiko    f    100   100    100
%
```

●プログラムファイル prog.awk

```
% cat prog.awk ⏎
BEGIN {
        total1=0
        total2=0
        total3=0
        sum=0
        n=0
        f=0
        printf "input file = students.dat¥n"
       }
       {
        total1+=$3
        total2+=$4
        total3+=$5
        n++
        print $1,$3+$4+$5
        if ($2=="f") {
                        f++
                        }
       }
END {
        printf "Ave1=" total1/n
        printf "¥nAve2=" total2/n
        printf "¥nAve3=" total3/n
        printf "¥nfemale=" f
        printf "¥n"
       }
%
```

このファイルを使って各科目の平均点や人数などを集計します。処理結果は次のようになります。

```
% awk -f prog.awk < infile⏎
input file = infile
Susumu 270
Kenichi 270
Masako 270
tarou 270
Reiko 300
Ave1=92
Ave2=94
Ave3=90
female=2
%
```

1・10

プログラム開発

UNIX に最もかかわりの深いプログラミング言語が

C 言語

です。UNIX では、C 言語のほかにも pascal、FORTRAN77、LISP をはじめとして数多くのプログラミング言語を利用することができますが、UNIX 自身も C 言語で記述されているという面からも、C 言語でのプログラミングが一般的です。UNIX で提供されている豊富なソフトウェアツールもほとんど C 言語でのプログラミングされています。

C 言語は

関数

を基本モジュールとし、構造化しやすい制御構造を記述することができるプログラミング言語です。

このほかにも、豊富なデータ型、演算子、制御構造が使えることも柔軟性の高いプログラミングを可能にしています。アセンブリ言語に比べれば極めて高水準な言語として使用することができますが、UNIX という OS を記述するシステム記述言語として必要な、アドレス・ポインタや構造体が使える低水準の機能を有するという二面性をもっています。

UNIX は、元来プログラム開発を強力に支援するツールを豊富に備えた OS です。エディタ、言語コンパイラをはじめとして通常必要とされるものは、そのほとんどが標準装備されています。

C 言語の処理系（コンパイラ）は

cc コマンド

によって利用することができます。また、コンパイル後はこの過程で作成された

オブジェクトファイル

を使ってリンクエディットしますが、これには

ld コマンド

を使います。ただし、一般には cc コマンドが自動的に ld コマンドを呼び出しますから、通常はユーザが特別にコマンド入力しなくても済んでしまいます。

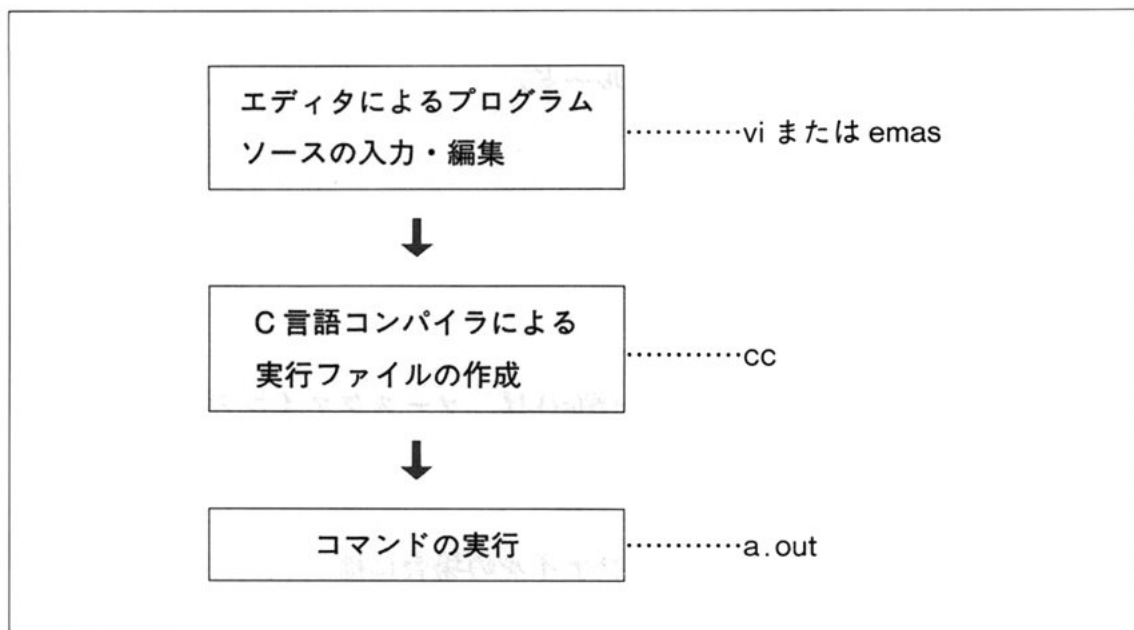
また、コンパイルに先だってプログラムソースのファイルを作成しますが、これには

テキストエディタ

が必要です。UNIX では、すでに説明した標準に提供されている vi エディタは

vi コマンド

を使って起動します。

●プログラム開発の手順

コンパイル、リンク(cc)と実行(a.out)

テキストエディタを使って入力したC言語のソースコードをマシン語のコードに変換させるためにコンパイルを行います。これには、

cc コマンド

を使います。C 言語コンパイラの出力のファイルは、オブジェクトコードで記述されていますが、これは通常、このままでは実行させることができません。他のオブジェクトプログラムやライブラリと連結させて、実行可能なファイルを

作成します。これを行うには、リンケージエディタの

ld コマンド

を使いますが、ld コマンドは cc コマンドから自動的に呼び出して利用することもできます。

実は cc コマンドでは、大ざっぱにいえば

プリプロセッサ

コンパイラ

リンケージエディタ（リンカ）

の一連のコマンドを呼び出して、ソースコードファイルを実行ファイルに翻訳する作業を行っています。

プリプロセッサの役割は、コンパイルを行う前に、

ファイルの取り込み（インクルード）

単純マクロ変換

条件付きコードの変換

などを行うことです。

cc コマンドの入力書式は、

cc［*options*］*files*

です。引数に指定する入力ファイル (*files*) は、ソースファイルの場合にはファイル拡張子に

.c

を付けたものを、またオブジェクトファイルの場合には

.o

を付けたものを指定します。

オプションに何も指定しない場合には、コンパイルとリンクが続けて実行されて、実行ファイルが作成されます。

```
% cat testfile.c ⏎
main()
{
        printf("My name is Susumu Funamoto¥n");
}
% cc testfile.c ⏎            ←──── コンパイル、リンクを実行する
% ls ⏎
```

```
a.out             testfile.c
% a.out ⏎                          ←―――― 実行ファイルを実行する
My name is Susumu Funamoto
%
```

実行ファイルは、

a.out

という名前で作成されますから、実行する際にはプロンプトに続けてa.outを入力します。

この実行ファイルに好みの名前を付けるには、

-o *file* オプション

を指定します。このオプションの直後に指定した名前 (*file*) の実行ファイルが作成されます。

```
% cc -o testfile testfile.c ⏎ ←―――― 実行ファイルに testfile
% ls ⏎                               という名前を付ける
a.out            testfile            testfile.c
% testfile ⏎
My name is Susumu Funamoto
%
```

プリンタ出力(lpr)

プログラム開発を行っている過程では、エディタを使って入力・編集したソースファイルをプリンタに出力することはよくあることです。UNIXの場合、同時に複数のユーザの出力要求を処理しなければなりません。そこで、出力要求によって転送されてくるファイルを、一時的に

キュー

につないで管理しています。キューは、先入れ先出し方式の待ち行列ですから、古いものから順に出力されるようになっています。

このキューにつながれたプリントジョブは

スプーラ

というプログラムによって順番に出力されるという仕組みになっています。

プリンタへの出力要求は、ファイルのデータをスプーラ管理下のキューに転送することによって実行されます。

lpr ［*options*］ *files*

出力対象のファイル(*files*)を引数に指定します。この際に出力されるプリンタは、通常は各プリンタに付けられている名称のなかで、

lp

がデフォルト値として選ばれます。この出力先を変更するためには、

-P オプション

を付けます。

```
% lpr -Plp01 exprog.c ⏎
```

プリンタ名lp01のプリンタに、ファイルexprog.cを出力します。

SVR系のUNIXでは、プリンタへの出力要求lpコマンドの書式は

lp ［*options*］ *files*

となります。

キューに繋がれたプリントジョブの状態を知るときには、

lpq コマンド

を使います。

lpq ［*jobs*］

引数には、対象となるプリントジョブの番号(*jobs*)を指定します。省略したときには、キューにあるすべてのプリントジョブの状態を出力します。

```
% lpq ⏎
lp is ready and printing
Rank      Owner     Job  Files       Total Size
active    funamoto  257  exprog.c    282 bytes ← 出力中の
                                                 プリントジョブ
1st       funamoto  203  exprog10.c  401 bytes ┐ 出力待ちの
2nd       Takeda    204  exfil       55 bytes  ┘ プリントジョブ
%
```

キューの管理下にあるプリントジョブの取消を行うには、

lprm コマンド

を使います。

lprm [*jobs*]

引数には、対象となるプリントジョブの番号（*jobs*）を指定します。これは、lpq コマンドで出力される内容の中の Job の項目の番号です。キューにあるすべてのプリントジョブの出力を取り消すときには、

-

を引数に指定します。

```
% lpq ⏎
lp is ready and printing
Rank       Owner     Job  Files       Total Size
active     funamoto  257  exprog.c    282 bytes
1st        funamoto  203  exprog10.c  401 bytes
2nd        Takeda    204  exfile      55 bytes
% lprm 203 ⏎ ←――― ジョブ番号 203 を取り消す
% lpq ⏎
lp is ready and printing
Rank       Owner     Job  Files       Total Size
active     funamoto  257  exprog.c    282 bytes
1st        Takeda    204  exfile      55 bytes
%
```

コンパイル手順の自動化（make）

小規模なユーティリティをプログラミングするなどの特別な場合を除いて、通常のプログラム開発では1つのプログラムをいくつかのモジュールに分割するのが効率的な方法です。大規模なプログラムが1つのファイルに記述されていれば、たった1カ所の修正でもプログラム全体をコンパイルしなくてはなりませんから、コンパイル時間や修正個所の発見に非常に効率の悪い作業を強いられます。また、分担作業をするという点からも、1つの大きなプログラムファイルの作成は避けなければなりません。

ところが、モジュールに分割してソースプログラムを作成するときには、逆に

・モジュール間の依存関係の管理

・バージョンアップの管理

などがわずらわしい問題になってきます。

そこで、これらの作業を自動化するために、

make コマンド

が使われます。

make コマンドでは、開発に関係するファイルやコマンドを

makefile（または Makefile）

という名前のファイルに記述しておけば、必要最小限のコンパイルおよびリンクを自動的に行って、間違いなく最新版の実行ファイルを作成してくれます。

```
a.out : main.o,sub.o
<tab>  cc main.o sub.o
```

この makefile の記述は、a.out が main.o と sub.o をリンクすることによって作成されることを記述しています。例えば、sub.o のソースファイル sub.c が a.out の作成以降に更新されているとすれば、

```
% make ⏎
```

と入力すれば、自動的に sub.c をコンパイルし、sub.o を作成し、a.out が他のオブジェクトファイルつまり main.o との再リンクによって作成され直されます。

デバッグ(dbx)

テキストエディタを使ってソースプログラムを入力し、コンパイラに通して実行ファイルを作成してからこれを実行したときに、そのまま即思いどおりに動作することは、プログラムの規模が大きくなればなるほどめったにありません。何らかのバグ（虫）がプログラムに入り込んでいるわけです。

このような場合の基本的な態度としては、もう一度ソースプログラムをチェックして誤りの個所を見つけることです。ところがこれではなかなか見つからない場合も少なくありません。

こういったときに、デバッグの作業を強力に支援してくれるツールが

デバッガ

です。UNIX では

adb

sdb

dbx

などのデバッガが使えます。ここでは、dbx の機能を中心に説明します。

dbx は、プログラムを実行させながらさまざまな機能を働かせて、プログラムが動作中の各種情報を入手することができる対話型のデバッガです。基本的な機能としては、

- 変数の値をソースプログラム中の名前で参照できる
- 1 ステートメント単位で実行を制御することができる
- プログラム中に中断点（ブレークポイント）を設定し、そこで実行を中断させることができる
- プログラムの動作経路や変数の値の変化を追跡できる
- デバッグ中にソースリストを表示できる

などがあります。

dbx ［*option*］*file*

dbx を使うためには、まずその準備として cc コマンドを使ってプログラムをコンパイルする際に、

-g オプション

を指定し、デバッグ時に必要なシンボルテーブルを作っておかなければなりません。

```
% cc -g -o exprog exprog.c ⏎
```

これで、実行形式ファイル exprog がデバッグオプション付きで作成されることになります。これをデバッグするために dbx を起動するには

```
% dbx exprog ⏎
Reading symbolic information...
Read 50 symbols
(dbx)  ←———— dbx のプロンプトが表示される
```

と入力すれば、dbx の各種機能を働かせるためのサブコマンドのモードに入ることができます。(dbx) は dbx のプロンプトで、これに続けてサブコマンドを

入力します。サブコマンドの種類とその使い方は、第2章のコマンドリファレンス dbx (p.172) を参照してください。

1・11

X ウィンドウシステム(X Window System)

UNIX を利用するハードウェア環境としては、ワークステーションが最も一般的な形態となってきています。そのため、従来はキャラクタ端末経由でコマンド入力を主体としていた操作環境から、マルチウィンドウ環境で、テキストだけでなくグラフィックスも扱える環境が使えるようになってきています。

UNIX 上でのウィンドウ環境を実現するシステムとしては、

X Window System (MIT)

NeWS (サンマイクロシステムズ)

Andrew (CMU)

などがありますが、事実上は X がほぼ標準ウィンドウシステムとして各メーカーのワークステーションに搭載されています。

X は、

クライアントサーバ

という方式を使って実行されているので、ネットワークを経由したプロセス間通信を行って、ユーザが直接使用しているマシンばかりか、別のマシンの資源をも活用した処理をすることができるようになっています。

ネットワークを経由したログイン機能 (rlogin) を使えば、他のマシンの機能や資源を利用することができますが、この場合は、あくまで文字情報に限られています。しかし、ネットワークの中に、例えばスーパーコンピュータがあってここで解析された結果を図形にして、UNIX ワークステーションに表示させたいときにも、X を使えばグラフィックス情報を容易に取り扱えるようになります。

もちろん、マルチウィンドウシステムですから、基本的に

1つの画面が複数の端末を同時に使用するのと同じ

コマンドを使わなくてもメニューによって操作が可能

などの利点によってユーザインタフェースが向上するのです。

Xの起動から終了まで

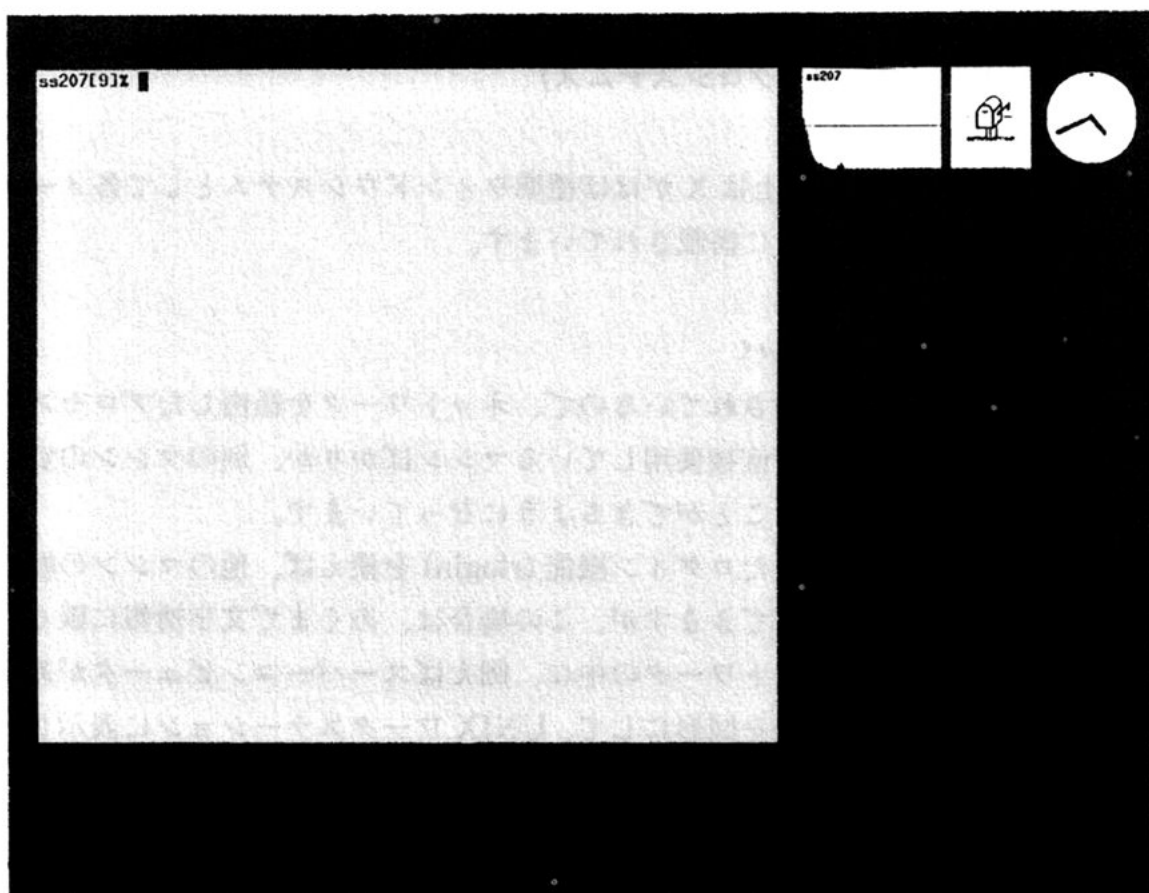
Xを起動するには

　　xinit コマンド

を使います。xinitは、Xのサーバを起動してさらに

　　xterm

と呼ばれるターミナルエミュレータを起動し、ディスプレイ上のウィンドウから、通常の端末と同じ操作ができる環境を提供します。もちろんこの時点でシェルが起動していますから、これまで説明したUNIXのコマンド操作は、このウィンドウでそのまま行えます。

●Xの起動時画面

Xが正常に起動すれば画面表示例のように、画面左上部にターミナルエミュレータのウィンドウが表示され、この中の左上には、プロンプトも表示され、さらにX字型の形状のマウスカーソルが表示されるはずです。なお、ターミナ

ルエミュレータは、複数起動させることができますが、このX起動時に表示されるウィンドウだけは、ログインウィンドウという特別な呼び方をします。

ターミナルエミュレータでのコマンド入力には、マウスカーソルがこのウィンドウの内部に位置して、I字型になっていなければなりません。

最終的にXを終了するには、ログインウィンドウ上でシェルを終了する方法をとれば同時に終了するようになっています。具体的には、プロンプトのところで CTRL + D または logout を入力します。

ウィンドウマネージャとウィンドウ操作

xinitでXを起動したときに出現するウィンドウは、本来ウィンドウ環境であればできるはずのウィンドウの移動、大きさ変更、アイコン化などがまったくできません。これを行えるようにするには、

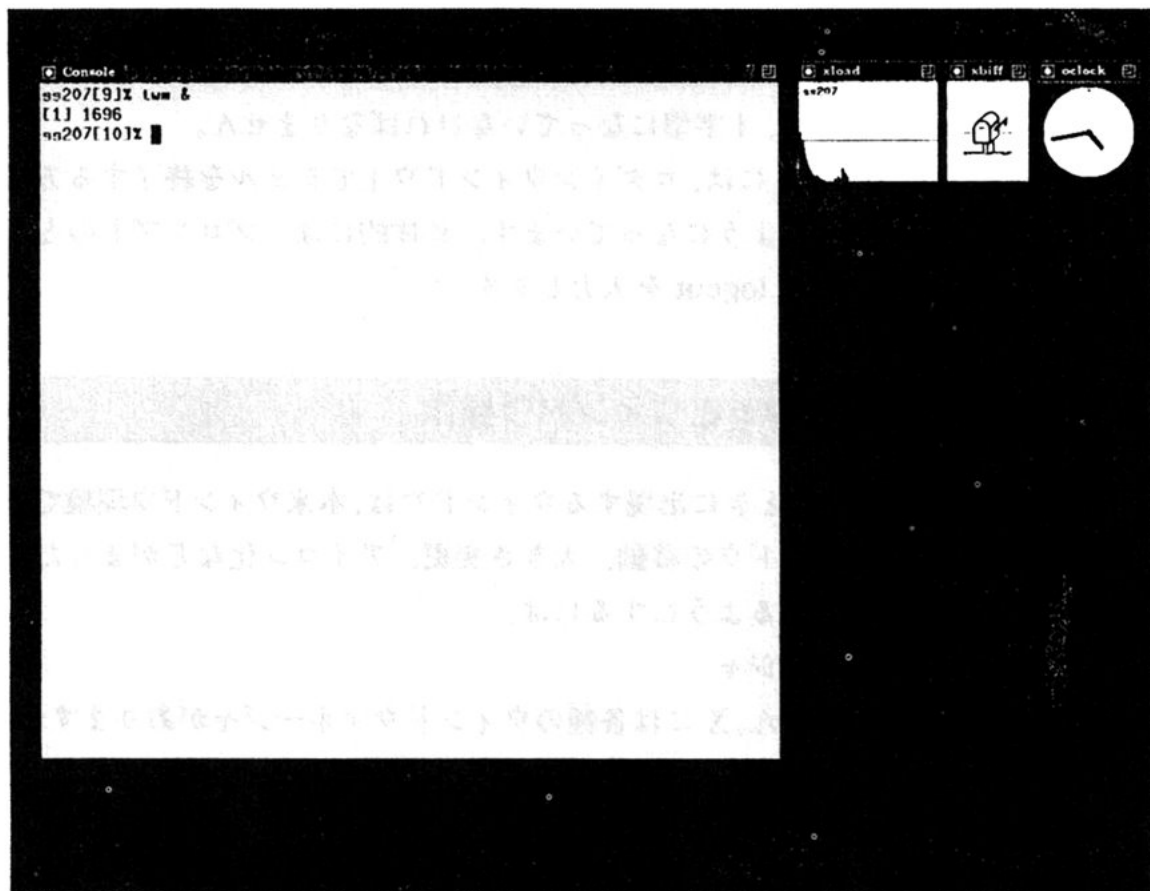
ウィンドウマネージャ

を使わなければなりません。Xには各種のウィンドウマネージャがありますが、ここでは

twm

を起動してみます。

ここでまず注意が必要なのは、ウィンドウマネージャは必ずバックグラウンドで実行させるために、&を付けることです。

● twm の起動時画面

twm によって表示されたウィンドウには、先ほどとは違ってウィンドウ上部に

　　タイトルバー

と呼ばれる部品が現れています。さらには、このタイトルバーの左端にアイコン化ボックス、右端にリサイズボックスがあります。これらは、それぞれマウスの左ボタンでクリックすれば、ウィンドウをアイコンにしたり、ウィンドウの大きさを変更することができます。

また、ウィンドウの外側の

　　ルートウィンドウ

という部分でマウスをクリックすれば、ポップアップメニューが表示されます。このメニューには、twm のウィンドウに指示する各種のコマンドが配置されていているので、これを使ってウィンドウ操作を行うことができます。

●メニュー上の主要コマンド機能一覧

Iconify	ウィンドウをアイコン化する
Rsize	ウィンドウの大きさを変更する
Move	ウィンドウを移動する
Raise	指定したウィンドウを一番手前にする
Lower	指定したウィンドウを一番奥にする
Focus	キーボード入力できるウィンドウを設定する

このようなウィンドウ自体に対する操作のほかにも、ウィンドウ間でその中のテキストを一方から他方へと簡単に張り付けることができます。まず、対象となるテキストをドラッグしてこの部分を反転させます。さらに張り付け先にマウスカーソルを移動させ、ここでマウスの中央のボタンを押します。

●張り付けの例

クライアントプログラム

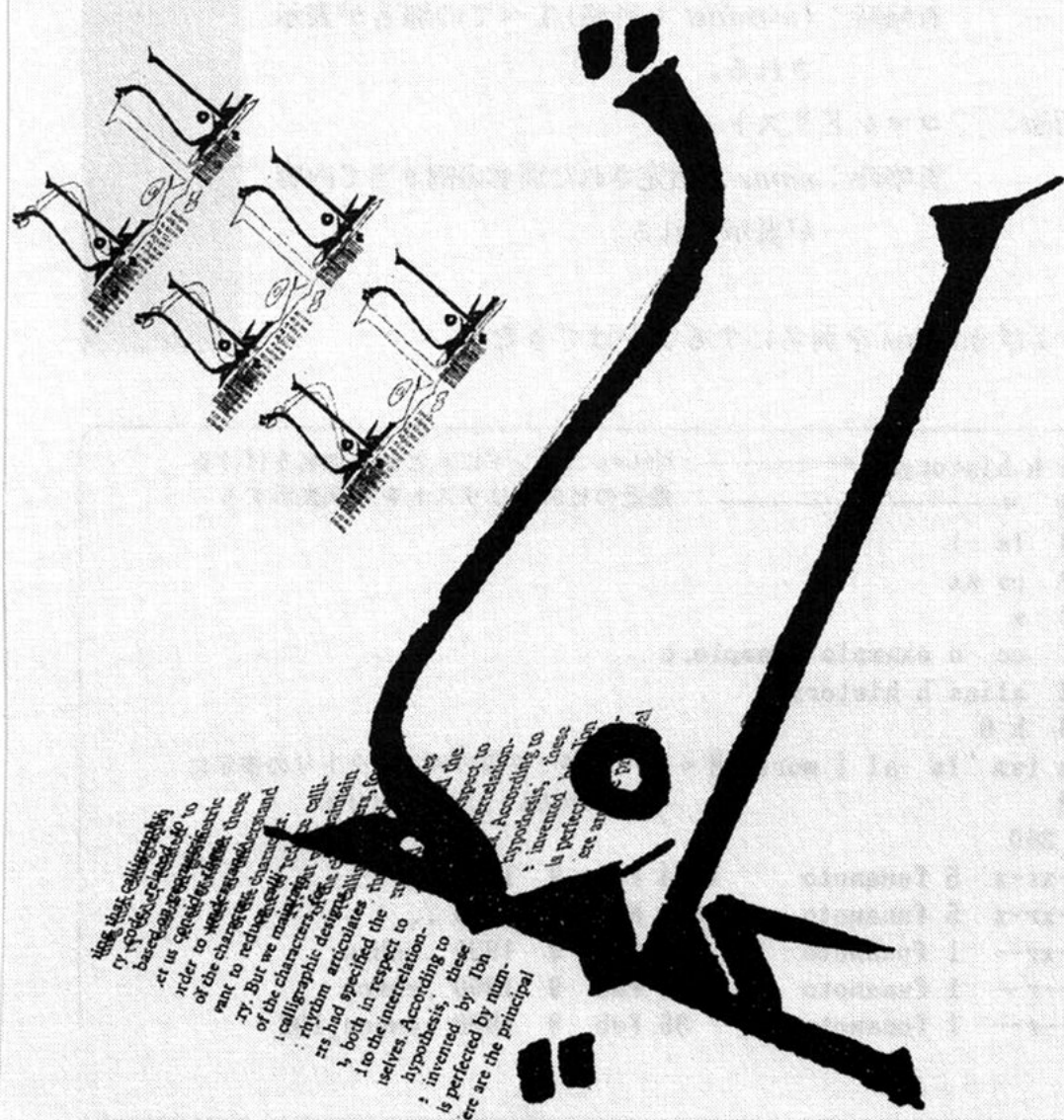
X上で動作することを意識して作成されたプログラムを

クライアント

と呼んでいます。標準的に提供されるクライアントにはけっこう役に立つものがあります。

xinit	：Xの初期化
twm	：ウィンドウマネージャ
xterm	：ターミナルエミュレータ
xclock	：時計
xload	：システムの負荷表示
bitmap	：ビットマップデータ作成ツール
xpr	：画面イメージの出力

第2章

コマンド・リファレンス

alias

コマンドに対する別名の割り当て

書式

alias [*name* [*wordlist*]]

●指定したコマンドのリスト（*wordlist*）に別名（*name*）を付ける。

【引数】

name　　：コマンドに対する別名

省略時：(*wordlist* も省略)すべての別名が表示される。

wordlist　：コマンドリスト

省略時：*name* で指定された別名の割り当て内容が表示される。

【注】

・alias および unalias を別名にすることはできない。

【使用例】

```
%alias h history⏎ ←────── history コマンドに h という別名を付ける
%h 6⏎              ←────── 最近のヒストリリストを 6 個表示する
    58  ls -l
    59  ps ax
    60  w
    61  cc -o example example.c
    62  alias h history
    63  h 6
%alias lsm 'ls -al | more'⏎ ←── 1 ページ毎のディレクトリの表示に
%lsm⏎                           lsm という別名を付ける
total 260
drwxr-xr-x  5 funamoto     1024 Feb  9  1990 .
drwxr-xr-x  5 funamoto      512 Nov  1 23:10 ..
-rwxr-xr--  1 funamoto      507 Feb  9  1990 .cshrc
-rw-r--r--  1 funamoto      554 Feb  9  1990 .emacs
-rw-r--r--  1 funamoto       38 Feb  9  1990 .emacs_102
```

```
-rwxr-xr-x  1 funamoto      436 Feb  9  1990 .history
-rwxr-xr--  1 funamoto      462 Feb  9  1990 .login
-rwxr-xr--  1 funamoto      127 Feb  9  1990 .profile
-rw-r--r--  1 funamoto        7 Feb  9  1990 .rhosts
-rwxr-xr-x  1 funamoto    24576 Jan 12 06:06 a.out
drwxr-xr-x  4 funamoto      512 Nov 14 07:09 cc
-rw-r--r--  1 funamoto  2188090 Oct 17 03:28 core
-rwxr-xr-x  1 funamoto    24576 Oct 29 04:45 exfile
-rw-r--r--  1 funamoto      296 Nov  2 03:04 exfile.c
-rw-r--r--  1 funamoto       30 Oct 28 23:27 exfile.dat
-rw-r--r--  1 funamoto       58 Oct 29 04:15 exfile.txt
-rwxr-xr-x  1 funamoto    24576 Oct 17 03:22 exfile1
-rw-r--r--  1 funamoto      290 Oct 25 05:34 exfile1.c
-rw-r--r--  1 funamoto      989 Oct 30 00:22 exfile1.o
-rw-r--r--  1 funamoto       58 Oct 28 23:18 exfile1.txt
-rw-r--r--  1 funamoto      282 Oct 17 07:33 exfile2.c
 --More--
-rw-r--r--  1 funamoto       44 Oct 28 23:24 exfile2.txt
-rwxr-x---  1 funamoto    24576 Jan 12 05:14 exfilea
-rw-r--r--  1 funamoto       40 Dec 13 04:55 exscript
-rwxr-xr-x  1 funamoto       67 Oct 30 00:32 infile
drwxr-xr-x  2 funamoto      512 Oct 30 02:36 link
drwxr-xr-x  2 funamoto      512 Oct 25 05:18 mk
-rw-r--r--  1 funamoto       40 Nov 14 06:58 outfileaa
-rw-r--r--  1 funamoto       27 Nov 14 06:58 outfileab
-rw-r--r--  1 funamoto       67 Oct 20 04:57 students.dat
-rw-r--r--  1 funamoto      453 Oct 20 04:59 total.awk
-rw-r--r--  1 funamoto       40 Nov 14 06:57 xaa
-rw-r--r--  1 funamoto       27 Nov 14 06:57 xab
%
%alias⏎ ←―――――――――――― すべての別名を表示する
^[[A     clrs100
back     set back=$old; set old=$cwd; cd $back; unset back; dirs
cd       set old=$cwd; chdir !*
em       emacs
h        history
lsm      ls -al | more
pd       pushd
pop      popd
print    print -T funamoto
pwd      echo $cwd
%
```

alias

at

時刻指定のコマンド実行

excute commands at a later time

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

at [*options*] [*time*] [*date*] [+*inc*] [*file*]

●指定した日時（*time*、*date*）に、指定されたファイル（*file*）に記述されたコマンド列を実行する。

【引数】

time：実行する時刻を指定
-l、-r オプション以外は必ず指定する。
1、2 または 4 桁の 10 進数を使う。
1、2 桁は時のみ、4 桁は時分と解釈する。
am（午前）と pm（午後）を付けることができる。
noon、midnight、now、next の文字列も指定できる。

date：日付
月名に日付を続けた文字列で記述する。

inc：指定した日時からの増加分 SUN PCU UXW
minutes、hours、days、weeks、months、year のいずれかを付けた数字も文字列で記述する。 HPU AIX OMR

file：実行するコマンド列を記述したファイル SUN NWS

【オプション】

-r *jobs*：指定した番号（*jobs*）のジョブのスケジューリングを取り消す。

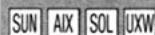
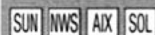
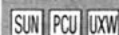
-l[*jobs*]：ジョブ（*jobs*）に関する情報を出力する。

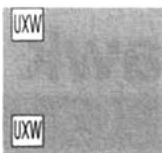
-c：C シェルを実行する。 SUN NWS AIX SOL

-s：B シェルを実行する。 SUN NWS AIX SOL

-j：日本語 C シェルを実行する。 NWS

-k：K シェルを実行する。 AIX

-m：メールを発信する。 SUN AIX SOL UXW

-f*file*　：ファイル（*file*）からコマンド別を読み込み実行する。

-d*job*　：指定したジョブ（*job*）の内容を表示する。

【使用例】

```
% at -c 0400pm exscript⏎ ←── 午後4時にCシェルスクリプトのexscript
job 2246 at Sat Oct 14 16:00:00 1989      を実行するように
                                          スケジューリングする
% at -l⏎ ←──── 現在スケジューリングされているジョブの情報を表示する
     2246 a    Sat Oct 14 16:00:00 1989

% at -r 2246⏎ ←──── ジョブ番号2246のスケジューリングを取り消す
% at -l⏎ ←──── スケジューリングされているジョブがないことを確認する
%
```

at

awk

パターン走査および簡易な処理をする言語

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

Aho, Weinberger, Kernighan

書式

awk [*option*] [*prog*] [*para*] [*files*]

●指定したテキストファイル（*files*）の中にあるパターン（*prog*）を捜す。抽出されたパターンに対して処理を行う。

【引数】

files　：パターンを捜す対象のテキストファイル名
　省略時または - の指定：標準入力から読み込む。

prog　：パターンのプログラム
　単一引用符（'）で囲んで指定する。

para　：コマンド行で使う変数の初期値
　形式：*valiable* = *value*
　　　（変数）　（値）

【オプション】

-f*file*　：指定したファイル（*file*）の内容をパターンとして使用する。

-F*c*　：指定した文字(*c*)をフィールド区切り文字とする。
　省略時：空白またはタブ区切り文字とする。

【注】

・*prog* に記述するプログラム構造とコマンド等は次のとおり。

<プログラムの構造>

```
BEGIN{
        program        } 初期処理を記述
        }                （実行は最初に1回だけ）
```

{
program ｝一致したパターンをすべて処理（繰り返し実行）
}

END {
program ｝終了処理（実行は最後に1回だけ）
}

（BEGINとENDのプログラムは省略できる）

<内部コマンド>

if（条件）文［else文］
while（条件）文
for（式；条件；式） 文
break
continue
{［文］...}
変数＝式
print［式リスト］［式］
printf書式［，式リスト］［ >式］
next この入力行に関して残りのパターンをスキップする。
exit 入力の残りの部分をスキップする。

<プログラム（*prog*）の一般的な書式>

ptn1{*action1*}
ptn2{*action2*}
⋮
ptnn{*actionn*}

<主なシンボル>

NF ：フィールドの数
NR ：入力行の数
FS ：フィールド区切り文字
RS ：入力行の区切り文字
OFS ：出力フィールドの区切り文字

ORS　　　　：出力行の区切り文字
FILENAME　：入力ファイル名

＜制御文の書式＞

・条件分岐（if 文）

if (*cond*) {
　sentence1
}
[else {
　sentence2
}]
	条件 cond が成立するときに文 *sentence1* を実行する。
	成立しないときに文 *sentence2* を実行する。
	else 以降はなくともよい。

・繰り返し（for 文）

for (*init*;*test*;*update*) {
　sentence
}
	式 *test* が真の間は文 *sentence* を繰り返し実行する。
	init は開始前に 1 回だけ、*update* は繰り返すたびに実行される。

・繰り返し（while 文）

while (*test*) {
　sentence
}
	式 *test* が真の間は文 *sentence* を繰り返し実行する。

・ジャンプ（break 文、continue 文、next 文、exit 文）

break	繰り返しを中断させる
continue	次の繰り返しから処理を続ける
next	次のレコードに処理をスキップする
exit	レコードの処理を終了させる（END は処理する）

・印刷（print 文、printf 文）

print	書式なしの印刷
printf	C 言語の printf 文と同じ機能

<組み込み関数>

exp(*var*)　：変数のもつ値の指数部分

int(*var*)　：変数のもつ値の整数部分

length(*var*)　：変数のもつ文字列の文字数

log(*var*)　：変数のもつ値の対数

sqrt(*var*)　：変数のもつ値の平方根

substr(*str, i, j*)　：変数のもつ文字列（*str*）の中で *i* 番目から *j* 番目の連続した文字列

<演算子>

・論理演算子　&&　かつ（and）

　||　または（or）

　!　〜でない（not）

・関係演算子　<　より小さい

　<=　より小さいか等しい

　==　等しい

　!=　等しくない

　>=　より大きいか等しい

　>　より大きい

・代入演算子　=　代入する

・増分演算子　+=　変数に指定した分だけ増加させる。

　++　変数の内容を 1 増加させる。

awk

<awkコマンドの完全正規表現に使う特殊文字>

パターン	意味
.	任意の1文字に一致
¥	特殊文字の機能を取り消し、通常文字として扱う
*	この直前の文字またはグループの0回以上の繰り返しに一致
^	行の先頭と一致
$	行の末尾と一致
+	この直前の文字またはグループの1回以上の繰り返しに一致
?	この直前の文字またはグループの0または1回以上の繰り返しに一致
[*str*]	文字列(*str*)のうちの1文字に一致
[*chr1-chr2*]	2つの文字(*chr1,chr2*)の範囲の1文字に一致
[^*str*]	文字列(*str*)のうちの文字以外の文字に一致
(expr)	正規表現のグループ
expr1 | expr2	正規表現 *expr1* または正規表現 *expr2*

【使用例】

ディレクトリの表示情報から、ファイル名(8番目の項目)とファイルサイズ(4番目の項目)の一覧を表示する。

```
%ls -l | awk '{print $8,$4}'⏎
a.out 24576
cc 512
core 2188090
exfile 24576
exfile.c 282
exfile1 24576
exfile1.c 282
exfile2.c 282
exfile4.c 281
infile 282
students.dat 67
total.awk 453
```

ファイルexfile.cの中の行から'char'という文字列を含む行をすべて表示する。

```
%awk '/char/ {print $0}' exfile1.c⏎
        while((c=getchar())!=EOF){
                        putchar(c);
                                putchar(c);
```

（注）exfile.cファイルの内容は、第1章を参照。

学生の成績ファイルstudents.datを使って各学生の3科目の合計点、各教科の平均点、女子の人数を求める。

```
%cat total.awk⏎
BEGIN {
        total1=0
        total2=0
        total3=0                        } 初期設定
        sum=0
        n=0
        f=0
        printf "input file = students.dat¥n"
      }
      {
        total1+=$3
        total2+=$4                      } 各科目の合計を加算する
        total3+=$5
        n++ ←──────────────────────── 学生数を加算する
        print $1,$3+$4+$5 ←─────────── 各学生の3科目の合計点を表示する
        if ($2=="f") { ←───────────── 女子の場合に加算する
                        f++
                        }
      }
END   {
        printf "Ave1=" total1/n
        printf "¥nAve2=" total2/n       } 各科目の平均を計算し表示する
        printf "¥nAve3=" total3/n
        printf "¥nfemale=" f ←──────── 女子の数を表示する
        printf "¥n"
      }
```

awk

```
%cat students.dat⏎ ←———— 学生の成績ファイルの内容を表示する
a m 80 95 60
b f 20 50 100
c m 40 40 40
d m 100 10 60
e f 20 60 90
%awk -f total.awk < students.dat⏎ ←— awk プログラム total.awk に対して
input file = students.dat⏎              入力を students.dat として実行する
a 235
b 170
c 120
d 170
e 170                                   処理結果
Ave1=52
Ave2=51
Ave3=70
female=2
%
```

banner
花文字の作成

SVR
SUN HPU NWS AIX
EWS
PCU UXW

書式

banner *strings*

●指定した文字列（*strings*）を花文字に変換して出力する。

【引数】

strings ：10 文字以下の文字列
　　　　　複数の文字列を指定できる

【使用例】

batch

バッチジョブの起動

SVR
SUN HPU AIX
DEC EWS OMR
PCU UXW

書式

batch [*options*] [*files*]

●バッチジョブを起動する。このジョブはシステムの負荷が低下したときに実行される。

【引数】

files ：実行するコマンド列を記述したファイル名　SUN
　　省略時：標準入力からコマンド列を読み込む。

【オプション】

-c ：C シェルを起動する。　SUN

-s ：B シェルを起動する。　SUN

-m ：メールを起動する。　SUN

【注】

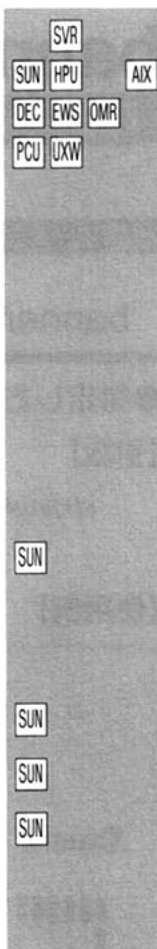
・at コマンド（p.120）を参照。

【使用例】

```
%batch⏎ ←―――――――――――― batch コマンドを実行する
at> sort < infile > outfile⏎ ←― バッチジョブとして sort コマンドを入力する
at> <EOT> ←―――――――――――― ^D で入力を終了する
job 2258 at Tue Jan  9 12:58:29 1990 ←―― スケジューリングされたジョブ
%                                          の情報
```

bc

任意精度の計算用簡易言語

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

bc ［*option*］［*files*］

● C 言語に似た構文で任意の精度の計算を会話形式で行う。ファイル(*files*)を指定すれば、この中に記述された計算を行う。dc コマンドのプロセッサ。

【引数】

files ：計算式を記述したファイル名

省略時：標準入力から読み込む。会話形式の計算ができる。

【オプション】

-c ：コンパイルだけを行う。

-l ：数値計算用ライブラリを使用する。

<bc コマンドで使える関数>

・オプションなし

（*e* を省略可能な符号と小数点付きの任意の桁数の数字とする）

sqrt(*e*) ：平方根

length(*e*) ：10 進数の有効桁数

scale(*e*) ：小数点以下の桁数

・-l オプション付き

s(*x*) ：正弦関数

c(*x*) ：余弦関数

e(*x*) ：指数関数

l(*x*) ：対数関数

a(*x*) ：逆正接関数

j(*n*,*x*) ：ベッセル関数

＜演算子＞

+ - * /	：加減乗除
%	：剰余
^	：べき乗
++ --	
== <= >= != < >	
= =+ =- =* =? =/ =^	

＜文＞

if 文

while 文

for 文

break 文

quit 文

defile 文

【使用例】

```
% bc⏎
5*10⏎ ←———— 5×10を計算する
50
x=10⏎
y=7⏎
x+y⏎          } x=10,y=7の代入をし、x+y,x÷yを実行する
17
x/y⏎
1 ←———— 整数値の範囲で計算される
^D ←———— インタプリタを終了する

% bc -l⏎ ←———— 数値計算ライブラリを使ってインタプリタを立ち上げる
x=10⏎
y=7⏎
x/y⏎
1.42857142857142857142 ←———— 正確な値（小数）が求められる
sqrt(y)⏎ ←———— 平方根の関数sqrtを使う
2.64575131106459059050
for (i=0;i<=7;i++) sqrt(i)⏎ ←———— 0から7までの自然数の平方根を求める
0
1.00000000000000000000
1.41421356237309504880
1.73205080756887729352
2.00000000000000000000
2.23606797749978969640
2.44948974278317809819
2.64575131106459059050
^D
%
```

bc

bg

バックグラウンド・ジョブの実行

background

書式

bg [%*job*]
bg [*pids*]

●指定した番号（*pids*）のジョブをバックグラウンドで実行する。

【引数】

pids　：プロセス番号
　　　　省略時：カレントのジョブを対象とする。

%job　：ジョブ番号

【注】

・jobs, fg, stop の各コマンドも参照のこと。

【使用例】

```
%cc exfile1.c⏎
^Z ←――――――――――――――――― 実行を中断する
Stopped
%bg⏎ ←――――――――――――――――― バックグラウンドで実行開始
[2]    cc exfile1.c &
%
[2]    Done                    cc exfile1.c ←――――― 終了メッセージ

%
```

biff

メールの着信の即時通知

BSD
SUN NWS AIX
DEC
UXW

書式

biff ［*option*］

●メールが着信したその時点でメッセージを出力する。

【オプション】

省略時：現在の状態を出力する。

y：着信の通知をする。

n：着信の通知をしない。

【使用例】

```
% biff ⏎      ←――――― 着信通知の状態確認
n             ←――――― 着信通知なし状態
% biff y ⏎    ←――――― 着信通知ありに設定する
%
```

cal

カレンダーの出力

calendar

SVR BSD SUN HPU NWS AIX DEC EWS OMR PCU UXW

書式

cal [[*month*] *year*]

●指定した西暦年（*year*）のカレンダーを出力する。

【引数】

year　：西暦年（1～9999）

省略時：現在の年月を対象とする。

month：月（1～12）

【使用例】

```
% cal 12 1989⏎ ←―――――― 1989年12月のカレンダーを表示する

   December 1989
 S  M Tu  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

```
% cal 1990⏎ ◀────── 1990年のカレンダーを表示する



                                 1990

             Jan                      Feb                      Mar
   S  M Tu  W Th  F  S     S  M Tu  W Th  F  S     S  M Tu  W Th  F  S
      1  2  3  4  5  6                 1  2  3                 1  2  3
   7  8  9 10 11 12 13     4  5  6  7  8  9 10     4  5  6  7  8  9 10
  14 15 16 17 18 19 20    11 12 13 14 15 16 17    11 12 13 14 15 16 17
  21 22 23 24 25 26 27    18 19 20 21 22 23 24    18 19 20 21 22 23 24
  28 29 30 31             25 26 27 28             25 26 27 28 29 30 31

             Apr                      May                      Jun
   S  M Tu  W Th  F  S     S  M Tu  W Th  F  S     S  M Tu  W Th  F  S
   1  2  3  4  5  6  7           1  2  3  4  5                    1  2
   8  9 10 11 12 13 14     6  7  8  9 10 11 12     3  4  5  6  7  8  9
  15 16 17 18 19 20 21    13 14 15 16 17 18 19    10 11 12 13 14 15 16
  22 23 24 25 26 27 28    20 21 22 23 24 25 26    17 18 19 20 21 22 23
  29 30                   27 28 29 30 31          24 25 26 27 28 29 30

             Jul                      Aug                      Sep
   S  M Tu  W Th  F  S     S  M Tu  W Th  F  S     S  M Tu  W Th  F  S
   1  2  3  4  5  6  7              1  2  3  4                       1
   8  9 10 11 12 13 14     5  6  7  8  9 10 11     2  3  4  5  6  7  8
  15 16 17 18 19 20 21    12 13 14 15 16 17 18     9 10 11 12 13 14 15
  22 23 24 25 26 27 28    19 20 21 22 23 24 25    16 17 18 19 20 21 22
  29 30 31                26 27 28 29 30 31       23 24 25 26 27 28 29
                                                  30
             Oct                      Nov                      Dec
   S  M Tu  W Th  F  S     S  M Tu  W Th  F  S     S  M Tu  W Th  F  S
      1  2  3  4  5  6                 1  2  3                       1
   7  8  9 10 11 12 13     4  5  6  7  8  9 10     2  3  4  5  6  7  8
  14 15 16 17 18 19 20    11 12 13 14 15 16 17     9 10 11 12 13 14 15
  21 22 23 24 25 26 27    18 19 20 21 22 23 24    16 17 18 19 20 21 22
  28 29 30 31             25 26 27 28 29 30       23 24 25 26 27 28 29
                                                  30 31


%
```

calendar

スケジュールの確認

SVR BSD
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

書式

calendar

●ホームディレクトリにあるファイル calendar に記述された予定を、その前日および当日に出力する。

【使用例】

```
% cat calendar ⏎        ←―――― ファイル calendar の内容を出力する
8/18 My birthday ⏎
% date ⏎
Sun Aug 18 14:22:16 JST 1993
% calendar ⏎
8/18 My birthday        ←―――― 設定するメッセージが出力される
%
```

【注】

・ファイル calendar に記述する各行には日付が必要。その形式には、

8/18

Aug. 18

August 18

が使用できる。また月の指定にアスタリスク*を使えば、毎月その日が対象になる。

cancel

ラインプリンタへの出力要求の取り消し

SVR
HPU NWS AIX
EWS OMR SOL
PCU

書式

cancel [*options*] [*ids*] [*prts*]

●指定した出力要求ID（*ids*）またはプリンタ（*prts*）の要求を取り消す。

【引数】

ids　：出力要求ID

prts　：ラインプリンタ名

【オプション】

-a　：すべての出力要求を削除する。　HPU

-e　：スプールキューを空にする(スーパーユーザのみ)。　HPU

-i　：ローカル要求だけを削除する。　HPU

-*user*　：指定したユーザ（*user*）の出力要求を削除する。　HPU

【注】

・IDおよびプリンタ名は、lpstatコマンドで知ることができる。

・プリンタへの出力要求は、lpコマンドを使う。

cat

ファイルの連結または出力

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

concatenate

書式

cat [*options*] [*files*]

●指定したファイル(*files*)を順番に読み込んで標準出力に書き出す。

【引数】

files ：ファイル名
　省略時または - の指定：標準入力から読み込む。

【オプション】

-n ：行番号を付ける。 SUN NWS AIX SOL
　-b ：-n オプションと同時に指定すれば、空行以外に行番号を付ける。

-s ：連続した複数の空行を1行にして出力する。 SUN NWS AIX SOL
　ファイルが存在しなくてもメッセージは出力しない。 HPU EWS OMR PCU UXW

-u ：バッファリングをしない。

-v ：非印字文字(タブ、改行、用紙送りを除く)を
　^X、M-x
　などのように、識別可能な形式で出力する。
　-vオプションと同時に指定するものとして以下のオプションがある。
　-e ：行末に$を付ける。
　-s ：読み込みができないファイルについてもメッセージは出力しない。 SUN
　-t ：タブを^Iで出力する。用紙送りを^Lで出力する。

【使用例】

```
%cat exfile1.txt⏎ ←―――― テキストファイル exfile1.txt の内容を表示する
unix
 SunOS   Sun Microsystems
 NEWS-OS SONY
 PC-UX/V NEC
%cat exfile2.txt⏎ ←―――― テキストファイル exfile2.txt の内容を表示する
dos
 PC-DOS   IBM
 MS-DOS   NEC,FUJITSU,...

%cat exfile1.txt exfile2.txt⏎ ←―― exfile1.txt と exfile2.txt の内容を結合して
unix                                表示する
 SunOS   Sun Microsystems
 NEWS-OS SONY
 PC-UX/V NEC
dos
 PC-DOS   IBM
 MS-DOS   NEC,FUJITSU,...
%cat -n exfile1.c⏎ ←―――― exfile1.c の内容を行番号を付けて表示する
     1  #include <stdio.h>
     2  #define DEL1 '¥010'
     3  #define DEL2 '¥138'
     4  main()
     5  {
     6          int c,p;
     7          p='¥000';
     8          while((c=getchar())!=EOF){
     9                  if (c!=DEL1 && c!=DEL2){
    10                          putchar(c);
    11                  } else {
    12                          if (
    13                              (p>=129 && p<=159)
    14                              || (p>=224 && p<=252)){
    15                                  putchar(c);
    16                          }
    17                  }
    18                  p=c;
    19          }
    20          sub();
    21          exit(0);
    22  }
```

```
%cat > exfile.dat⏎ ←—— 標準入力からのデータを exfile.dat に書き出す
aaaaa
bbbbb
ccccc
^D ←—————————— データの入力を終了する
%cat exfile.dat⏎ ←—— exfile.dat の内容を表示して確認する
aaaaa
bbbbb
ccccc
%cat >> exfile.dat⏎ ←—— 標準入力からのデータを exfile.dat に追加して書き出す
ddddd
eeeee
^D
%
%cat exfile.dat⏎ ←—— exfile.dat の内容を表示して確認する
aaaaa
bbbbb
ccccc
ddddd
eeeee
%
```

cb

C言語のプログラムソースの整形

c program beautifier

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

cb [*options*] [*files*]

●指定したC言語のソースプログラム（*files*）を整形する。

【引数】

files　：C言語のソースプログラムファイル名
　　　　省略時：標準入力から読み込む。

【オプション】

省略時：入力されたソースの改行をそのまま使用する。

-s　　：Kernighan & Ritchieの「THE C Programming Language」(日本訳、共立出版：「プログラミング言語C」）で使われている標準的な形式で出力する。　SUN HPU OMR SOL / AIX EWS PCU UXW

-j　　：分割された行を接続する。　SUN HPU OMR SOL / EWS AIX PCU UXW

-l *leng*：指定した長さ（*leng*）で行を分割する。　SUN HPU OMR SOL / EWS AIX PCU UXW

【使用例】

```
% cat infile.c⏎ ←――― C言語のソースリスト infile.c の内容を表示する
#include <stdio.h> ←――― 以下はネスティングのない見にくいソースリスト
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
int c,p;
p='¥000';
while((c=getchar())!=EOF){
if (c!=DEL1 && c!=DEL2){
putchar(c);
} else {
```

```
if (
(p>=129 && p<=159)
|| (p>=224 && p<=252)){
putchar(c);
}
}
p=c;
}
exit(0);
}
% cb infile.c > outfile.c⏎ ←── infile.c の内容をフォーマッティングし、
                                 結果を outfile.c に出力する
% cat outfile.c⏎ ←────── outfile.c の内容を表示する
#include <stdio.h> ←───── 以下は整形されたソースになっている
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);

}
%
```

cc

C言語コンパイラ

c compiler

SVR BSD SUN HPU NWS DEC EWS OMR PCU UXW

書式

cc ［*options*］ *files*

●指定されたC言語のファイル（*files*）をコンパイルする。

【引数】

files：C言語のソースファイル名（拡張子c）
　または、オブジェクトファイル名(拡張子o)。

【オプション】（＊）

-c　：コンパイルのみを行う。
　拡張子にoの付いたオブジェクトファイルが生成される。

-g　：デバッガ用の付加情報を生成する。
　dbxコマンドの対象ファイルを作成する。　SUN NWS DEC
　sdbコマンドの対象ファイルを作成する。　EWS OMR PCU UXW
　cdb, xdbコマンドの対象ファイルを作成する。　HPU

-o *outfile*　：実行形式のファイル名（*outfile*）を指定する。
　省略時：a.outがファイル名になる。

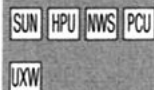
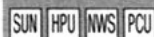
-p　：profコマンド用の情報ファイル(mon.out)を生成する。

-pg　：gprofコマンド用の情報ファイル（gmon.out）を生成する。　SUN NWS HPU

-w　：エラーの出力をしない。　SUN HPU NWS PCU

-O　：コンパイルコードの最適化をする。

-S　：アセンブリ言語ソースの出力をする。ファイル拡張子は、sとなる。

-C　：プリプロセッサによるコメントの削除を禁止する。　SUN HPU NWS PCU UXW

【使用例】

```
%ls -l⏎ ←――――― ワーキングディレクトリでのディレクトリ情報を表示する
total 1
-rw-r--r--  1 funamoto      282 Oct 15  1989 exfile1.c
%cc exfile1.c⏎                                    ↑
%ls -l⏎                                      C 言語のソースファイル
total 25
-rwxr-xr-x  1 funamoto    24576 Oct 15  1989 a.out ←―― 実行ファイル
-rw-r--r--  1 funamoto      282 Oct 15  1989 exfile1.c
%cc -o exifle exfile1.c⏎
%ls -l⏎
total 49
-rwxr-xr-x  1 funamoto    24576 Oct 15  1989 a.out
-rwxr-xr-x  1 funamoto    24576 Oct 15  1989 exfile1 ←―― 名前を付け
-rw-r--r--  1 funamoto      282 Oct 15  1989 exfile1.c   た実行ファイル
%cc -g -o exfile1 exfile1.c⏎ ←― デバッグオプションを付けてコンパイルする
%
```

◆デバッガの使用については、dbx コマンド（p.172）を参照。

cd

ワーキングディレクトリの移動および変更

change working directory

書式

cd ［*dir*］

●ワーキングディレクトリを指定したディレクトリ（*dir*）に変更する。

【引数】

dir ：新しいワーキングディレクトリ名

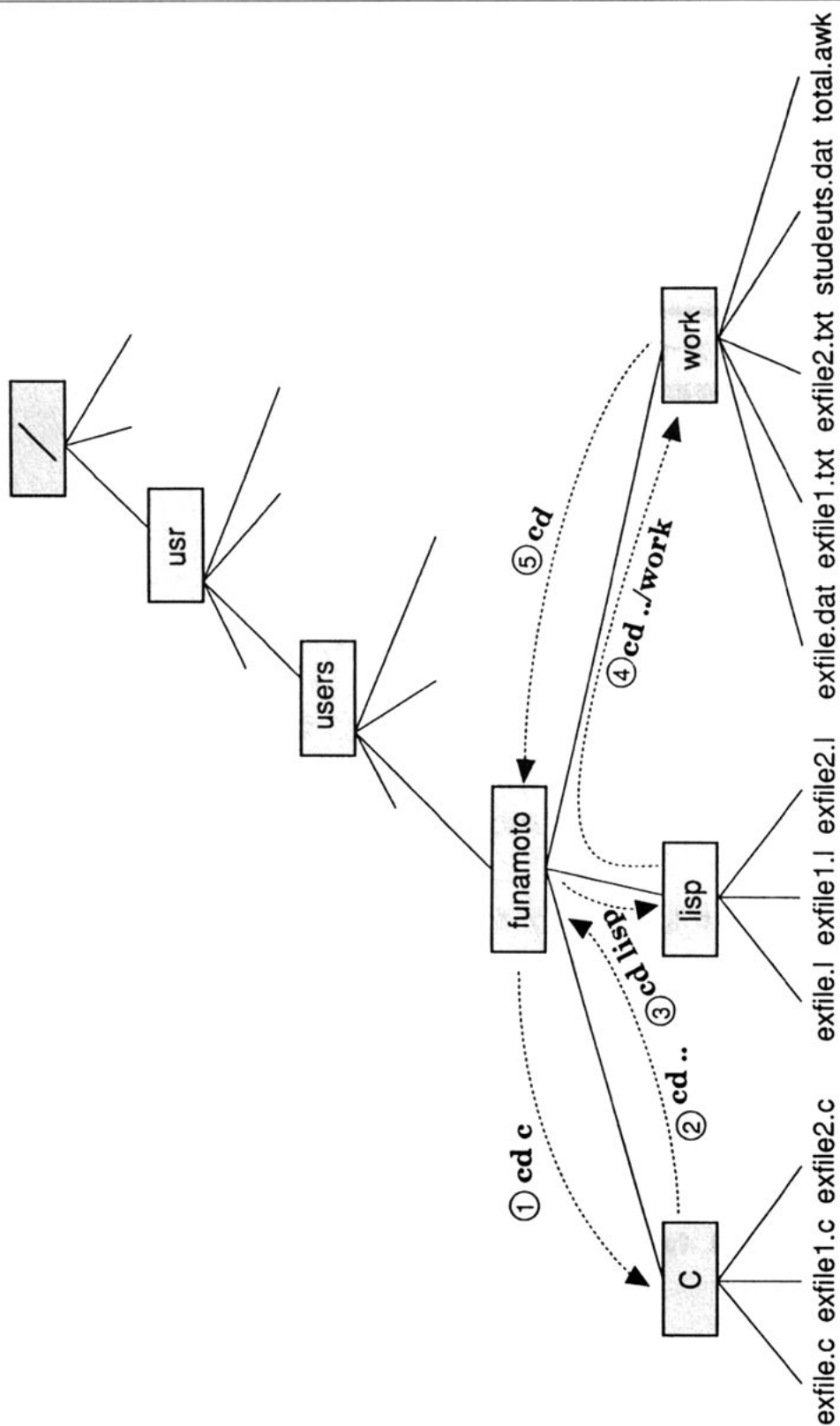
省略時：環境変数$HOMEの値をワーキングディレクトリとする。

cd

【使用例】

%pwd↵ ← 現在のワーキングディレクトリのパスを表示する
/usr/users/funamoto
%ls↵ ← ワーキングディレクトリでのディレクトリ情報を表示する
c lisp work ← c,lisp,work の各ディレクトリが存在する
%cd c↵ ← ①ワーキングディレクトリを c に移動する
%pwd↵
/usr/users/funamoto/c
%ls↵ ← ワーキングディレクトリでのディレクトリ情報を表示する
exfile.c exfile1.c exfile2.c exfile3.c
%cd .. ↵ ← ②ワーキングディレクトリを親（1 つ上）に移動する
%pwd↵
/usr/users/funamoto
%cd lisp↵ ← ③ワーキングディレクトリを lisp に移動する
%pwd↵
/usr/users/funamoto/lisp
%ls↵
exfile.l exfile1.l exfile2.l
%cd ../work↵ ← ④ワーキングディレクトリを親（1 つ上）のディレクトリにある work に移動する
%pwd↵
/usr/users/funamoto/work
work に移動していることを確認する
%ls↵
exfile.dat exfile1.txt exfile2.txt students.dat total.awk
%cd↵ ← ⑤ ワーキングディレクトリをホームディレクトリに戻す
%pwd↵
/usr/users/funamoto
%

cd

/
usr
users
funamoto
c
lisp
work
① cd c
② cd ..
③ cd lisp
④ cd ../work
⑤ cd
exfile.c exfile1.c exfile2.c
exfile.l exfile1.l exfile2.l
exfile.dat exfile1.txt exfile2.txt studeuts.dat total.awk

chgrp

ファイルのグループIDの変更

change file group

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

chgrp ［*options*］ *group files*

●指定されたファイル（*files*）のグループIDを別のグループID（*group*）に変更する。

【引数】

group ：グループID（グループ名でも可）

files ：ファイル名

【オプション】

-f ：エラー発生時でも何も出力しない。 SUN NWS AIX DEC SOL

-R ：ディレクトリを再帰的に下降して、すべてのサブディレクトリのファイルのグループIDを変更する。 SUN NWS AIX DEC SOL UXW

-h ：シンボリックリンクのグループを変更する。 UXW

【注】

・スーパーユーザもしくはファイルの所有者だけが使用できる。

・chownコマンド（p.153）を参照。

【使用例】

```
% ls -lg exfile⏎ ←――――― ファイルexfileのディレクトリ情報を表示する
-rw-r-----  1 funamoto   aigroup       147 Oct 30 16:46 exfile
% chgrp oagroup exfile⏎ ←― ファイルexfileのグループ名aigroupをoagroupに変更する
% ls -lg exfile⏎ ←――― 変更されていることを確認する
-rw-r-----  1 funamoto   oagroup       147 Oct 30 16:46 exfile
%
```

chmod

ファイルの許可モードの変更

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

change mode

書式

chmod [*options*] *mode files*

●指定したファイル(*files*)の許可モードを新たな許可モード(*mode*)に変更する。

【引数】

files ：ファイル名

mode ：許可モード（絶対値または記号による指定が可能）

<絶対値によるモード指定>

以下のモードの論理和によってできる8進数。

0400 所有者による読み取り

0200 所有者による書き込み

0100 所有者による実行

0040 グループによる読み取り

0020 グループによる書き込み

0010 グループによる実行

0004 その他のユーザによる読み取り

0002 その他のユーザによる書き込み

0001 その他のユーザによる実行

4000 実行時にユーザIDをセットする。

2000 実行時にグループIDをセットする。

1000 sticky ビットをオンにする。

<記号によるモード指定>

【書式】[ugoa] [+ | =] [rwxst]

u ユーザに対する許可

g グループに対する許可

o その他のユーザに対する許可

a　ugo のすべてが指定されたのと同義

＋　許可モードの追加

｜　許可モードの削除

＝　他のビットをすべてリセットする。

r　読み取り許可

w　書き込み許可

x　実行許可

s　セット所有者 ID またはセットグループ ID がオン

t　sticky ビットをオン（u とともに指定する）

l　アクセス中に必須ロック発生　OMR

W　ディレクトリかもしくは、他の実行ビットがセットされている時に実行許可をセットする。　SUN NWS

【オプション】

-f：モード変更が不可の場合でもメッセージ出力はしない。　SUN NWS AIX DEC

-R：再起的にディレクトリを下降して、すべてのファイルのモードを変更する。　SUN NWS AIX DEC UXW

【注】

・ファイルの所有者またはスーパーユーザだけが使用できる。

【使用例】

```
%ls -l exfile↵ ←―― ファイル exfile のディレクトリ情報を詳細に表示する
-rw-r-----  1 funamoto       147 Oct 30 16:46 exfile
   exfile は、所有者に読み出し/書き込み許可、グループに読み出し許可がある
%chmod g+w exfile↵ ← ファイル exfile に対してグループの書き込み許可を与える
% ls -l exfile↵
-rw-rw----  1 funamoto       147 Oct 30 16:46 exfile
%chmod g-w exfile↵ ← ファイル exfile に対してグループの書き込み許可を除く
%ls -l exfile↵
-rw-r-----  1 funamoto       147 Oct 30 16:46 exfile
% chmod 660 exfile↵ ← ファイル exfile に対してグループの書き込み許可を与える
                       (660＝400＋200＋40＋20)
%ls -l exfile↵
-rw-rw----  1 funamoto       147 Oct 30 16:46 exfile
%
```

chown

ファイルの所有者の変更

change file owner

SVR BSD HPU NWS AIX EWS OMR SOL PCU UXW

chown

書式

chown ［*options*］ *owner files*

●指定したファイル（*files*）の所有者を新たな所有者（*owner*）に変更する。

【引数】

owner：所有者名
10進のユーザIDまたはログイン名で指定可能。

files　：ファイル名

【オプション】

-f　：所有者変更が不可の場合でもメッセージ出力はしない。 NWS AIX SOL

-R　：再起的にディレクトリを下降して、すべてのファイルの所有者を変更する。 NWS AIX SOL UXW

-h　：シンボリックリンクの所有者を変更する。 UXW

【使用例】

所有者 (root)、グループ名 (oagroup)、全ユーザに対し読み出し/書き込み許可となっているファイル exfile を所有者(funamoto)、グループ名(aigroup)所有者に対する読み出し/書き込みとグループに対する読み出しに制限する設定を行う。

```
% ls -lg exfile⏎
-rw-rw-rw-  1 root      oagroup      147 Oct 30 16:46 exfile
% chown funamoto exfile⏎
% ls -lg exfile⏎
-rw-rw-rw-  1 funamoto  oagroup      147 Oct 30 16:46 exfile
% chgrp aigroup exfile⏎
% ls -lg exfile⏎
-rw-rw-rw-  1 funamoto  aigroup      147 Oct 30 16:46 exfile
% chmod 640 exfile⏎
% ls -lg exfile⏎
-rw-r-----  1 funamoto  aigroup      147 Oct 30 16:46 exfile
%
```

chsh

ログイン時に起動するシェルの変更

BSD
SUN HPU NWS AIX
DEC EWS
PCU

change default login shell

書式

chsh *name* ［*shell*］

●ログイン時に起動されるシェルを変更する。パスワードファイル（/etc/passwd）のログインシェル名が書き換えられる。

【引数】

name ：利用者のログイン名
（スーパーユーザのみ指定可能）

shell ：シェル名
Bシェルならば /bin/sh、Cシェルならば /bin/csh を指定する。
省略時：/bin/sh が指定されているものとする。

【使用例】

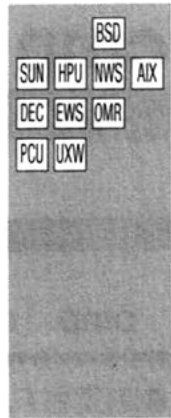
```
%cat /etc/passwd⏎
          .
          .
          .
user01:ntAulMS.05es2:100:200::/usr/users/user01:/bin/csh
user02:POTLBpzuqJ2XQ:101:200::/usr/users/user02:/bin/csh
user30:muxx:E.G0McT5HtXMg:102:200::/usr/users/user30:/bin/csh
user99:h0deET1jc0L7M:103:200::/usr/users/user99:/bin/csh
          .
          .
          .
%chsh /bin/sh⏎ ←── 自分（user30）のシェルを標準シェル（Bシェル）に変更する
%cat /etc/passwd⏎
          .
          .
          .
user01:ntAulMS.05es2:100:200::/usr/users/user01:/bin/csh
user02:POTLBpzuqJ2XQ:101:200::/usr/users/user02:/bin/csh
user30:muxx:E.G0McT5HtXMg:102:200::/usr/users/user30:/bin/sh
user99:h0deET1jc0L7M:103:200::/usr/users/user99:/bin/csh
          .
          .
          .
%
```

clear
画面クリア

clear terminal screen

書式

clear

●端末の画面をクリアする。

cmp

2つのファイルの内容比較

compare two files

書式

cmp ［*options*］ *file1 file2*

●指定された2つのファイル（*file1* と *file2*）の内容を1バイトごとに比較する。

【引数】

file1　：ファイル名
　　　　-を指定すれば標準入力から読み込む。

file2　：ファイル名

【オプション】

省略時：ファイル内容が同一の時は、メッセージの出力なし。
　　　　ファイル内容が異なるときは
　　　　・先頭からのバイト数（10進）
　　　　・内容（8進）
　　　　を出力する。

-l　　：異なるバイトごとに
　　　　・先頭からのバイト数(10進)
　　　　・内容(8進)
　　　　を出力する。

-s　　：メッセージの出力なし。終了コードが返却される。
　　　　0 ……ファイルの内容が同一
　　　　1 ……ファイルの内容に相違あり
　　　　2 ……ファイルのアクセス不能、引数なし、システムエラー

【使用例】

```
%cat exfile1.c⏎
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥138'
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        sub();
        exit(0);
}
%cat exfile2.c⏎
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
```

相違点 1

相違点 2

cmp

```
}

C言語のソースリスト exfile1.c と exfile2.c を比較して、相互の違いを表示する
%cmp exfile1.c exfile2.c⏎
exfile1.c exfile2.c differ: char 57, line 3 ← 3行目でファイルの先頭から
                                               57文字目が違っている
%echo $status⏎ ← 終了ステータスを表示する
1
%
```

【注】

・2個目の相違点は表示されない。

・cmpコマンドは、2つのファイルに相違があるかどうかを調べる場合に使われる。具体的な相違点が必要なときにはdiffコマンドを使う。

colrm

カラム(桁)の削除

BSD SUN NWS AIX DEC OMR

書式

colrm [*scol* [*ecol*]]

●標準入力から読み込んだデータの指定した範囲のカラムを標準出力には書き出さない。

【引数】

scol　：開始カラム数
　　省略時：先頭カラム (1)

ecol　：終了カラム数
　　省略時：行末

【使用例】

```
% cat exfile.dat ⏎
Funamoto Susumu     Tokyo      (3722)1234
Date Masamune       Miyagi     (77)4567
Takeda Shingen      Yamanashi  (34)8901
Funamoto Syotaro    Tokyo      (3333)4321
Date Tadamune       Sendai     (79)1357
Takeda Katsuyori    Suwa       (23)2468
% colrm 20 29 <exfile.dat ⏎←── 20～29 カラムの範囲を出力させない
Funamoto Susumu     (3722)1234
Date Masamune       (77)4567
Takeda Shingen      (34)8901
Funamoto Syotaro    (3333)4321
Date Tadamune       (79)1357
Takeda Katsuyori    (23)2468
%
```

colrm

comm

2つのファイルの共通行の選択または削除

SVR BSD
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

書式

comm ［*options*］ *file1* *file2*

●ソート済みの2つのファイル（*file1*,*file2*）を比較して、片方だけある行と両方にある行に分類する。

【引数】

file1,*file2*　：ファイル名

【オプション】

省略時	：1列目にfile1だけにある行、2列目にfile2だけにある行、3列目に両方にある行を出力する。
-1	：1列目を出力しない。つまりfile2にある行を出力する。
-2	：2列目を出力しない。つまりfile1にある行を出力する。
-3	：3列目を出力しない。つまり違いの行を出力する。
-12	：file2にだけある行を出力する。
-23	：file1にだけある行を出力する。
-123	：何も出力しない。

【使用例】

```
% cat exfile3 ⏎
Reijiro
Susumu
Syotaro
% cat exfile4 ⏎
Masako
Susumu
Syotaro
% comm exfile3 exfile4 ⏎
        Masako ←——— exfile 3 に存在しない行
Reijiro ←————————— exfile 4 に存在しない行
                Susumu  ┐
                Syotaro ┘←——— 相方のファイルに共通して存在する行
%
```

comm

cp

ファイルのコピー

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

copy

書式

❶ cp ［*options*］ *file1 file2*
❷ cp -r ［*options*］ *dir1 dir2*
❸ cp ［*options*］ *files dir*

❶指定したファイル（*file1*）の内容を別のファイル（*file2*）にコピーする。

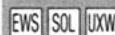
❷指定したディレクトリ（*dir1*）の内容やサブディレクトリを別のディレクトリ（*dir2*）にコピーする。

❸ *files* に指定した１対上のファイル（*files*）を指定したディレクトリ（*dir*）にコピーする。

【引数】

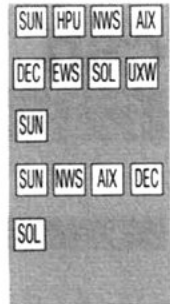
file1、*file2*：ファイル名
dir1、*dir2*：ディレクトリ名
dir2 で指定されたディレクトリが存在しないときには作成される。
files　：ファイル名
dir　：ディレクトリ名
すでに存在していなくてはならない。

【オプション】

-i　：コピーされるファイルがすでに存在するときに、確認のための入力要求をする。 SUN NWS AIX DEC EWS SOL UXW
y　……コピーを実行する。
y 以外　……コピーを中止する。
-p　：umask を無視したモードにする。 EWS
-f　：強制的に上書きする。 DEC

-r　：ディレクトリに対して再帰的にコピーする(書式❷、❸)。

-R　：ディレクトリに対して再帰的にコピーする(書式❷、❸)。

-p　：ファイルの修正時刻とモードを変更しない。

cp

【注】

・自分自身に対するコピーはできない。

【使用例】

```
% ls -l↵ ←―――― 現在のワーキングディレクトリの内容を表示する
total 1
-rw-r--r--  1 funamoto      282 Oct 17  1989 exfile1.c
% cp exfile1.c exfile2.c↵ ←― exfile1.c をコピーして exfile2.c を作成する
% ls -l↵ ←―――― exfile2.c が作成されていることを確認する
total 2
-rw-r--r--  1 funamoto      282 Oct 17  1989 exfile1.c
-rw-r--r--  1 funamoto      282 Oct 17  1989 exfile2.c ← 作成された exfile2.c
                                    ┌― exfile1.c をコピーして exfile2.c を作成する
                                    │  この際に exfile2.c が存在しているときには
                                    │  確認する
% cp -i exfile1.c exfile2.c↵ ←┘
overwrite exfile2.c? y↵ ←―――― exfile2.c に上書きしてよいかの確認
                                  メッセージ。y を入力して上書きする
% mkdir dir1↵ ←―――― 新たなディレクトリの dir1 を作成する
% cp *.* dir1↵ ←―――― ワーキングディレクトリの全ファイルをサブディレク
                        トリ dir1 にコピーする
% ls -l dir1↵ ←―――― dir1 のディレクトリの内容を表示する
total 2
-rw-r--r--  1 funamoto      282 Oct 17  1989 exfile1.c
-rw-r--r--  1 funamoto      282 Oct 17  1989 exfile2.c
%                 確かに exfile1.c と exfile2.c が dir1 にコピーされている
```

csh

Cシェルの起動

BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

a shell with C like syntax

書式

csh [*options*] [*args*]

● Cシェルを起動する。

【引数】

args ：コマンドの引数
-c オプション使用時のみ指定する。

【オプション】

-b ：このオプション以降のオプション指定を無視する。 SUN NWS AIX SOL

-c*cmd*：指定したコマンド（*cmd*）を実行する。
*args*で指定した内容はargvに保存される。

-e ：起動したコマンドが異常終了もしくは終了コードが0以外の時にシェルを終了する。

-f ：Cシェルを迅速に起動する。
.cshrcは実行しない。ログインシェルであれば、.loginも実行しない。

-i ：対話形式で処理を進める。

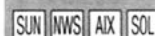
-n ：コマンドの解析だけを行う。実行はしない。

-s ：標準入力からコマンドを読み込む。

-t ：1行読み込んでコマンドを実行する。

-v ：verbose変数を設定する。変数の置換後にコマンドが表示される。

-V ：.cshrcの実行前にverbose変数を設定する。

-x ：echoの変数を設定する。変数の置換前にコマンド行が表示される。

-X　：.cshrc の実行前に echo 変数を設定する。

【注】

・詳細は第 3 章 C シェルリファレンスを参照。

【使用例】

```
%cat exscript⏎ ←── コマンドを記述したファイル（exscript）の内容を表示する
date ←──────────────── 日付の表示
cal 12 1989 ←───────── カレンダーの表示
repeat 3 echo funamoto ←── funamoto を 3 回表示
%csh exscript⏎ ←──────── exscript を C シェルで実行する
Tue Dec 12 14:57:06 JST 1989
   December 1989
 S  M Tu  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
funamoto
funamoto
funamoto
%
```

cut

ファイルからのフィールドの切り出し

書式

cut *options files*

●指定したファイル(*files*)からオプションで指定したフィールドを切り出す。

【引数】

files ：ファイル名

【オプション】

-c*list* ：文字位置の指定

-f*list* ：区切り文字で分けられたフィールドの指定

-d*chr* ：フィールド区切り文字（*chr*）の指定
省略時：タブ

-s ：フィールド区切り文字のない行を無視する。

【使用例】

```
% cat exfile ⏎
001:Susumu Funamoto:3722-1234   }
002:Syotaro Funamoto:3333-5678  } 例題ファイルの内容
003:Reiziro Funamoto:5555-4321  }
% cut -f2 -d: exfile ⏎  ←―― 第2フィールドを切り出す
Susumu Funamoto
Syotaro Funamoto
Reiziro Funamoto
% cut -c5-6 exfile ⏎  ←―― 5文字目から6文字目までを切り出す
Su
Sy
Re
%
```

【注】

・*list* は

　整数のフィールド番号をカンマで区切る昇順の並び

　整数のフィールド番号と-による範囲

を使える。

cut

date

日付の出力と設定

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

date [*options*] [*mmddhhmm* [*yy*]] [+*format*]

●日付・時刻の設定および出力をする。

【引数】

mmddhhmm [*yy*]：月日時分 [年]
（スーパーユーザのみ設定できる）
省略時：出力する

mm	月
dd	日
hh	時間（24時間制）
mm	秒
yy	年

省略時：現在の年

+*format*　：出力形式を指定する。　SUN HPU EWS AIX
指定がないときには *mmddhhmm* [*yy*] 形式。　DEC EWS OMR UXW

<フィールド記述子>

n	改行
t	タブ
m	月（01～12）
d	日（10～31）
y	年（00～99）
D	日付（mm/dd/yy）
H	時（00～23）
M	分（00～59）
S	秒（00～59）
T	時刻（HH：MM：SS）

j　　年間通算日（001～366）
w　　曜日（0～7）
a　　曜日（Sun～Sat）
h　　月（Jan～Dec）
r　　時刻（AM/PM法）

【オプション】

-u	：GMT（グリニッジ標準時）で設定および出力を行う。	SUN NWS DEC SOL UXW
-a ［-］*sss.fff*	：時刻を秒単位（*sss*）で調整する。 -（マイナス）によって指定もできる。 *fff* は秒の端数。	SUN SOL UXW SUN
-n	：LAN上のマシンでの同期をとらない。	NWS AIX OMR
-c	：UCT（Coordinated Universal Time）を使う。	DEC

【使用例】

```
%date⏎ ←――――― 現在の日時を表示する
Sat Oct 28 14:15:32 JST 1989
%date -u⏎ ←――――― グリニッジ標準時を表示する
Sat Oct 28 05:15:42 GMT 1989
%
```

date

dbx

プログラムのソースレベルのデバッガ

extended debugger

書式

dbx [*options*] [*objfile* [*corefile*]]

●指定された実行形式ファイル(*objfile*)をそれぞれの言語(C、Passcal、FORTRAN77など)のソースレベルでデバッグする。

【引数】

objfile ：デバッグ対象のオブジェクトファイル名
オブジェクトファイル内にシンボルテーブルが含まれるようにするために、コンパイル時に | g オプションを指定しなければならない。
省略時：a.out が指定されたものとする。

corefile ：*corefile* ファイル名が指定された場合には、そのプログラムがエラーを起こした時の状態を調べることができる。
省略時：core が指定されたものとする。

【オプション】(*)

-r ：デバッグモードのプロンプトを表示せずに、*objfile* を即刻実行する。

-i ：標準入力が端末装置であるかのように作動する。 NWS DEC EWS OMR

-k ：カーネルをデバッグする。 SUN NWS AIX DEC

-I*dir* ：ソースファイルをサーチするディレクトリのリストに *dir* を追加する。

-c *file* ：標準入力から読み取りを行う前に、*file* に収められている dbx のコマンドを実行する。 NWS EWS AIX DEC SOL

-s *file* ：*file* に納められている初期化コマンドを実行する。 SUN

<dbx コマンドで使える主なサブコマンド>

サブコマンド	機 能
assign *var=val*	変数(*var*)に値(*val*)を代入する。
cont	実行を再開する。
delete *n*	指定した番号(*n*)のトレースやブレークポイントの設定を解除する。
file	ファイルの内容を表示する。
func	関数の内容を表示する。
help	サブコマンドの一覧を表示する。
list *l1*, *l2*	指定した範囲（*l1*〜*l2*）のソースリストを表示する。
next	次の行を実行する。
print *exp*	指定したパターン（*exp*）の変数の値を表示する。
run	実行を開始する。
sh *cmd*	コマンド(*cmd*)の実行
source *file*	指定したファイル(*file*)からコマンドを読み込む。
status	トレースやブレークポイントの設定状況を表示する。
step	次の行を実行する。
stop in *proc* stop at *line* stop *var*	指定した関数(*proc*)にブレークポイントを設定する。 指定した行(*line*)にブレークポイントを設定する。 指定した変数(*var*)にブレークポイントを設定する。
trace *line* trace *proc* trace *var*	指定した行(*line*)、関数(*proc*)、変数(*var*)をトレースする。
quit	*dbx* を終了する。
whatis *var*	指定した変数(*var*)が宣言されている部分を表示する。
where	関数の呼出状況を表示する。
whereis *var*	指定した変数(*var*)が使われている部分を表示する。

【注】

・dbx コマンドでは、起動する際の初期設定ファイル

.dbxinit

を使うことができます。alias, stop などのサブコマンドをこのファイルに記述しておけば、毎回起動時に自動的に実行されます。

【使用例】

C言語のソースプログラムexfile1.cをデバッグオプション付きでコンパイルし、このオブジェクトプログラムexfile1についてデバッガdbxを使ってその動作を確認する。

```
% cc -g exfile1.c -o exfile1⏎
% dbx exfile1⏎ ←――――― exfile1に対してdbxを起動する
Reading symbolic information...
Read 50 symbols
(dbx) help⏎ ←――――― dbxで使用できるコマンドの一覧を表示する
     Command Summary

Execution and Tracing
  catch     ignore    run       stop
  clear     next      status    trace
  cont      rerun     step      when
  delete

Displaying and Naming Data
  assign    dump      undisplay where
  call      print     up        whereis
  display   set81     whatis    which
  down

Accessing Source Files
  cd        func      pwd       /
  edit      list      use       ?
  file

Miscellaneous
  alias     detach    make      sh
  dbxenv    help      proc      source
  debug     kill      quit

Dbxtool
  button    toolenv   unbutton  unmenu
  menu

Machine Level
  nexti     stepi     stopi     tracei
```

```
The command `help <cmdname>' provides additional
help for each command
(dbx) list 1,21⏎ ←――――― 1〜21 行のソースを表示する
    1   #include <stdio.h>
    2   #define DEL1 '¥010'
    3   #define DEL2 '¥137'
    4   main()
    5   {
    6           int c,p;
    7           p='¥000';
    8           while((c=getchar())!=EOF){
    9                   if (c!=DEL1 && c!=DEL2){
   10                           putchar(c);
   11                   } else {
   12                           if (
   13                               (p>=129 && p<=159)
   14                               || (p>=224 && p<=252)){
   15                                   putchar(c);
   16                           }
   17                   }
   18                   p=c;
   19           }
   20           exit(0);
   21   }
(dbx) stop at 9⏎ ←――――― 9 行目にブレークポイントを設定する
(1) stop at "/usr/users/funamoto//exfile1.c":9
(dbx) run < infile > outfile⏎ ←―― 標準入力に infile,標準出力に outfile を割
Running: exfile1 < infile > outfile   り当てプログラムを実行する
stopped in main at line 9 in file "exfile1.c" ←―― 9 行目で実行中断
    9                   if (c!=DEL1 && c!=DEL2){
(dbx) print c⏎ ←――――― ここで変数 c の内容を表示して確認する
`exfile1`main`c = 128  ┌― 現在のトレース、ブレークポイント、関数呼び出しの
(dbx) status⏎ ←――――――┘  状態を表示する
(1) stop at "/usr/users/funamoto/exfile1.c":9 ←┘ 9 行目にブレークポイントが設定されている
(dbx) delete 1⏎ ←―― この設定を解除する。表示された行の先頭の番号を指定する
(dbx) status⏎ ←―― 解除されていることを確認
(dbx) cont⏎ ←―― 実行を再開する

execution completed, exit code is 0
program exited with 0
(dbx) quit⏎ ←―― デバッガを終了する
%
```

dd

ファイルのフォーマット変換およびコピー

書式

dd *options*

●指定した入力ファイルをフォーマット変換したうえで出力する。

【オプション】

if=*file*	：入力ファイル名 省略時　標準入力
of=*file*	：出力ファイル名 省略時　標準出力
ibs=*n*	：入力のブロックサイズ（バイト数） 省略時　*n*=512
obs=*n*	：出力のブロックサイズ（バイト数） 省略時　*n*=512
bs=*n*	：入出力のブロックサイズ（バイト数） ibs、obs より優先指定される。
cbs=*n*	：変換用バッファサイズ conv オプションで ascii、ebcdic、ibm、block、unblock を指定した場合に必要。
skip=*n*	：変換前にスキップするレコード数
files=*n*	：終了前に入力ファイルをコピーする数　SUN NWS AIX DEC SOL UXW
seek=*n*	：変換前にシークする先頭からのレコード数
count=*n*	：変換するレコード数

conv=ascii	：EBCDIC → ASCII
ebcdic	：ASCII → EBCDIC
ibm	：ASCII → EBCDIC
block	：可変長→固定長

unblock ：固定長→可変長

lcase ：アルファベットを小文字に変更

ucase ：アルファベットを大文字に変更

swab ：対になっているバイトを交換

noerror ：エラー発生後も処理続行

sync ：ibs に入力の全レコードをすべて詰める。

これらの変換指定は(,)で区切って複数指定可。

【使用例】

```
%cat infile⏎
a m  80  95  60
b f  20  50 100
c m  40  40  40
d m 100  10  60
e f  20  60  90
%dd if=infile of=outfile conv=ucase⏎ ←── infileの中の小文字を大文字に変換する
0+1 records in
0+1 records out
%cat outfile⏎
A M  80  95  60
B F  20  50 100
C M  40  40  40
D M 100  10  60
E F  20  60  90
%
```

deroff

nroff, troff 構文の除去

delete roff expression

書式

deroff ［*options*］［*files*］

●指定したファイル（*files*）中に記述された roff 系の構文をすべて取り除き、この残りの部分を標準出力に書き出す。

【引数】

files ：ファイル名

【オプション】

-m*chr* ：*chr*=m のときは、mm マクロ定義行を削除する。
chr=l のときは、mm マクロ定義行に加え、mm リストを削除する。
chr=s のときは、ms マクロ定義行を削除する。

-w ：1 行に 1 ワードのリストを出力する。これ以外は削除する。

【使用例】

```
% cat extext ⏎
.TL
Sample text for NROFF
.AU
Susumu Funamoto
.AI
Nippon Electronics Collage
Artificial Intelligent Laboratory
.AB
This is a sample text.
It will help for you to understand what is "NROFF"
.AE
.NH
Introduction
.PP
Nroff formats test in the named files for typewriter-like device.
Options may appear in any order so long as they appear before the files.
% deroff extext ⏎

                        Sample text for NROFF

                          Susumu Funamoto

                      Nippon Electronics Collage
                   Artificial Intelligent Laboratory

                        This is a sample text.
          It will help for you to understand what is "NROFF"


Introduction

        Nroff formats test in the named files for typewriter-like
device. Options may appear in any order so long as they appear before
the files.
%
```

deroff

df

ファイルシステムに関する情報の出力

SVR BSD
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

disk free

書式

df [*option*] [*filesystems*] [*files*]

●マウントされているファイルシステム（*filesystems*）の空き領域などの情報を出力する。

【引数】

filesystems ：ファイルシステム名
　省略時：現在マウントされている全ファイルシステム

files ：ファイル名またはディレクトリ名。
　このファイルまたはディレクトリが存在するファイルシステムが対象となる。

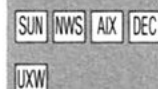
【オプション】

-a ：大きさが0ブロックのものを含めて、すべてのファイルシステムを対象とする。

-i ：使用および未使用のｉノード数を表示　SUN NWS AIX DEC UXW

-t *type* ：ファイルシステムのタイプ
(nfs、4.2 または 4.3 が指定可能)　SUN NWS UXW

-t ：空きブロック、空きｉノード数、使用済みブロック、使用済みｉノードを出力する。　HPU OMR PCU

-f ：フリーリスト内の正確な値を出力する。　HPU OMR PCU

【注】

・PC-UX/V ではスーパーブロック内のカウントを調べる。

【使用例】

```
%df -t 4.2⏎ ←── 4.2BSD タイプのファイルシステムの情報を表示する
Filesystem          kbytes    used   avail capacity  Mounted on
/dev/sd0a             7608    4237    2610    62%    /
/dev/sd0g            65845   55773    3487    94%    /usr
/dev/sd0e            31948   31289       0   109%    /export/swap
/dev/sd0d            14741    5671    7595    43%    /export/root
/dev/sd0h           163982   25905  121678    18%    /home
%
```

Filesystem　：ディスクのパーティション名
kbytes　：各パーティションの総量（キロバイト）
used　：使用済領域の量　（キロバイト）
avail　：未使用領域の量　（キロバイト）
capacity　：使用率　(%)
Mounted on　：マウントされているファイルシステム名

df

diff

2つのテキストファイルの差異の出力

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

differential file comparator

書式

diff [*options*] *file1 file2*
diff [*options*] *dir1 dir2*

●指定した2つのファイル(*file1* と *file2*)、または2つのディレクトリ(*dir1* と *dir2*)にあるファイル同士を比較し、相違点を出力し同一にするための情報を出力する。

【引数】

file1、*file2*：比較するファイル名
ファイル名が - のときには標準入力から読み込む。

dir1、*dir2*：比較するディレクトリ名
同じ名前をもつファイル同士を比較する。

【オプション】

-l　：空白文字タブを無視する。　SUN NWS DEC SOL PCU

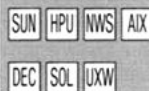
-c [*n*]　：指定した行数 (*n*) を出力する。　SUN NWS AIX DEC SOL UXW
省略時：n＝3

-D*string*：*file1* と *file2* の内容をマージしたテキストを標準出力に書き出す。Cプロセッサの制御については、*string* を指定しないときに *file1*、指定したときに *file2* がコンパイル対象となるように生成される。　SUN NWS AIX DEC SOL UXW

-e　：*file1* を *file2* に一致させるための ed エディタ用のコマンド列を出力する。　SUN HPU NWS AIX DEC SOL UXW

-f　：-e と逆順の出力をする。ed エディタでは使えない。

-h　：高速処理を行う。結果は必ずしも正確ではない。

diff

-i　　：大文字と小文字の違いを無視する。 SUN AIX DEL SOL UXW

-n　　：-e と逆順の出力をする。各挿入または削除コマンドについて、変更された行のカウントが出力される。 SUN NWS AIX DEC EWS OMR SOL PCU UXW

-b　　：後続ブランク（スペースとタブ）を無視する。

-w　　：ブランクとタブをすべて無視する。 SUN NWS DEC SOL PCU UXW

-t　　：出力行でタブを展開する。 SUN NWS DEC SOL PCU UXW

-C*n*　　：指定した行数（*n*）を出力する。 DEC

【オプション】（ディレクトリ比較時）

-l　　：ロングフォーマットで出力する。

-r　　：共通のサブディレクトリに対して再帰的に適用する。

-s　　：相違のないファイルの場合も報告する。

-S*name*　：ディレクトリの比較を指定したファイル（*name*）から開始する。

【注】

・3つのファイルの差異を出力するには diff3 コマンドを使う。

diff

【使用例】

C言語のソースリストexfile1.cとexfile2.cを比較して両者の違いから、同一にするための処理を出力する。

```
%diff exfile1.c exfile2.c
3c3
< #define DEL2 '¥138'
---
> #define DEL2 '¥137'
20d19
<         sub();
%
```


◆ exfile1.cとexfile2.cのソースリストはcmpコマンドの使用例を参照。

dircmp

ディレクトリの比較

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

directory comparison

書式

dircmp［*options*］*dir1 dir2*

●指定した2つのディレクトリ（*dir1* と *dir2*）にあるファイルを比較し、この結果を出力する。

【引数】

dir1、*dir2*：ディレクトリ名

【オプション】

-d ：両方のディレクトリに存在する同一の名前のファイルを比較する。
相違点がある場合には、これを diff コマンドと同一の形式で出力する。

-s ：一致したファイルについてのメッセージの出力をしない。

-w*n*：出力行の幅を *n* にする。
省略時：*n*＝72

【使用例】

```
% dircmp -d dir1 dir2⏎ ←—— ディレクトリ dir1 と dir2 のファイルを比較する

Nov 14 12:57 1989  Comparison of dir1 dir2 Page 1

directory        .
different        ./exfile.c ←———— exfile.c の内容は異なっている
same             ./exfile.txt ←———— exfile.txt の内容は同一
different        ./students.dat ←———— students.dat の内容は異なっている

Nov 14 12:57 1989  diff of ./exfile.c in dir1 and dir2 Page 1

4c4⏎
< #define DEL2 '¥136'     ┐
---                       ├ ./exfile.c の内容の相違点
> #define DEL2 '¥137'     ┘

Nov 14 12:57 1989  diff of ./students.dat in dir1 and dir2 Page 1

6c6⏎
< e f 20 60 90            ┐
---                       ├ ./students.dat の内容の相違点
> e f 20 60 100           ┘

%
```

◆相違点の出力形式については diff コマンド （p.182）を参照。

dirs

ディレクトリスタックの内容の出力

directory stack

CSH

書式

dirs [*option*]

●ディレクトリスタックの内容を出力する。出力内容は左側から順に最新。

【オプション】

-l：～(ホームディレクトリ)を使わない絶対パスで出力する。

【使用例】

```
%pwd↵
/usr/users/funamoto
%ls↵
ex1              ex2
%pushd ex1↵ ◄─────────── ディレクトリ ex1 を push する
~/ex1 ~
%pwd↵
/usr/users/funamoto/ex1
%ls↵ ◄─────────────── ワーキングディレクトリは ex1 に移動している
c       dir     roff
%pushd c↵ ◄──────────── ディレクト c を push する
~/ex1/c ~/ex1 ~
%dirs↵
~/ex1/c ~/ex1 ~
%ls↵
a.out          exfile.c       exfile1.c      exfile2.txt    students.dat
cc             exfile.dat     exfile1.o      infile         total.awk
core           exfile.txt     exfile1.txt    link
exfile         exfile1        exfile2.c      mk
%popd↵ ◄─────────────── ディレクトリスタックから pop する
~/ex1 ~
%pwd↵
/usr/users/funamoto/ex1
%popd↵
~

%pwd↵
/usr/users/funamoto
%
```

du

ディレクトリの使用状況の出力

SVR BSD
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

disk usage

書式

du [*option**s*] [*name**s*]

●ディレクトリまたはファイル単位でのディスクの使用状況を
　ブロック数
　キロバイト数
で出力する。

【引数】

names：ディレクトリ名またはファイル名
　　　省略時：ワーキングディレクトリ名

【オプション】

省略時　：ディレクトリのブロック数（キロバイト数）だけを出力。
-s　：ブロック数の総合計だけ出力。
-a　：各ファイルについてのブロック数を出力。
-r　：読み取り不可またはオープン不可のファイルについてメッセージを出力する。

SUN AIX OMR SOL
PCU

du

【使用例】

```
%pwd⏎ ←── 現在のワーキングディレクトリを表示する
/usr/users/funamoto/exdir
%du⏎ ←── ワーキングディレクトリ以下の各ディレクトリの
          ファイルの使用量を表示する
4       ./c/cdir
5       ./c
4       ./lisp
2       ./csh
8       ./work
3       ./dir1
1       ./exdir2
4       ./dir2
28      .
%
```

echo

引数の内容の出力

echo arnguments

CSH

SUN NWS OMR

書式

echo [*option*] [*args*]

●引数（*args*）に指定された文字列を標準出力に出力する。

【引数】

args　：出力される文字列
複数の文字列を指定するときには、空白またはタブで区切る。
省略時：標準入力から読み込む。

【オプション】

-n　：改行を出力しない。

【使用例】

```
%echo hello world⏎ ←―― hello world という文字列を表示する
hello world
%echo -n hello world⏎ ←―― hello world という文字列を表示する際に、
hello world%                 表示後の改行をしない
%echo $status⏎ ←―― 終了ステータスを表示する
0
%echo $history⏎ ←―― ヒストリリストの長さを示すシェル変数の
100                         内容を表示する
%echo *⏎ ←―― ワーキングディレクトリのファイル名を展開する
exfile.c exfile1.c exfile2.c exfile3.c
%echo ls *.c⏎ ←―― ワイルドカードを使ったコマンドを展開する
ls exfile1.c exfile2.c
%
```

ed

テキストエディタ

text editor

書式

ed [*options*] [*file*]

●テキスト・ラインエディタ ed を起動する。

【引数】

file ：編集対象のファイル名
　　省略時：テキストはバッファに格納される。

【オプション】

-	：エディタからのプロンプトや各種メッセージを出力しない。	AIX NWS OMR EWS DEC HPU
-p *string*	：プロンプト列（*string*）を指定する。	SOL AIX SUN PCU OMR EWS DEC HPU UXW
-s	：エディタからのプロンプトや各種メッセージを出力しない。	SOL AIX SUN PCU UXW

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

ed

【使用例】

```
%ed exfile.c⏎ ←――――― exfile.c を編集するために ed エディタを起動する
296
1,$p⏎ ←――――― 1行目から最終行（$）までを表示する
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
/* comment */
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
}
3p⏎ ←――――― 3行目を表示する
#define DEL2 '¥137'
s/7/6/p⏎ ←――――― この行の7を6に置換する
#define DEL2 '¥136'
4p⏎ ←――――― 4行目を表示する
/* comment */
4d⏎ ←――――― 4行目を削除する
4p⏎ ←――――― 4行目を表示する
main()
$p⏎ ←――――― 最終行を表示する
}
$a⏎ ←――――― 最終行の次に追加する
/* comment */⏎ ←――――― 追加行
.⏎ ←――――― 追加を終了する
w⏎ ←――――― ファイルに書き出す
296
```

ed

```
q⏎ ←───────── エディタを終了する
%cat exfile.c
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥136' ←───────── 置換されている
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                          if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
}
/* comment */ ←───────── 追加されている
%
```

ed

ex

edの上位互換のテキストエディタ

extended text editor

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

ex [*options*] *files*

●テキストエディタ拡張版 ex を起動する。

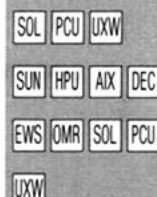
【引数】

files ：編集対象のファイル名

【オプション】

- \- ：エディタからのプロンプトや各種メッセージを出力しない。
- -v ：vi を起動する。
- -t *tag* ：タグ（*tag*）を含んだファイルを編集する。（このタグは、ctag コマンドで作成されたもの） SUN NWS AIX DEC OMR SOL PCU UXW
- -r *file* ：指定したファイル（*files*）を復旧する。
- -L ：クラッシュ時に保存したファイル名を出力する。 SOL PCU UXW
- -R ：読み込み専用モードにする。 SUN HPU AIX DEC EWS OMR SOL PCU UXW
- -l ：LISP 言語のプログラム用のモードにする。 SUN HPU AIX DEC OMR SOL
- -w*n* ：表示行数（*n*）を指定する。 SUN AIX SOL
- +*cmd* ：指定したコマンド（*cmd*）を実行してから編集開始する。 SUN NWS OMR EWS DEC EWS OMR
- -c *cmd* ：指定したコマンド（*cmd*）を実行し、編集開始する。 PCU UXW

【使用例】

```
%ex exfile.c⏎ ←────── exfile.c を編集するために ex エディタを起動する
"exfile.c" 22 lines, 296 characters
```

```
:%⏎ ←――――――――― 全リストを表示する
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥136'
/* comment */
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
}
:4⏎ ←――――――――― 4行目に移動して表示する
/* comment */
:4d a⏎ ←――――――― 4行目を削除してバッファaに格納する
main()
:$⏎ ←――――――――― 最終行に移動して表示する
}
:put a⏎ ←―――――― バッファaの内容を取り出す
/* comment */
:! ls⏎ ←―――――――― lsコマンドを実行する
"exfile.c" 22 lines, 296 characters
a.out       exfile.c     exfile1.c     exfile2.txt   students.dat
cc          exfile.dat   exfile1.o     infile        total.awk
core        exfile.txt   exfile1.txt   link
exfile      exfile1      exfile2.c     mk
!
:wq⏎ ←――――――――― ファイルに書き出してエディタを終了する
"exfile.c" 22 lines, 296 characters
%
```

ex

fg

フォアグラウンド・ジョブとしての実行

CSH

foreground

書式

fg [*pid*]
fg [%*job*]

●指定した番号（*pid*）のジョブをフォアグラウンドで実行する。

【引数】

pid ：プロセス番号
省略時：カレントのジョブが指定される。

%*job* ：ジョブ番号

【使用例】

```
%cc exfile.c &↵  ← C言語のプログラムのコンパイルをバックグラウンドジョブとして実行する
[1] 773  ← [1]：ジョブ番号　773：プロセス番号
%fg↵  ← カレントジョブをフォアグラウンドで実行する
cc exfile.c
%
```

file

ファイルの種類の検査と出力

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

file type

書式

file [*options*] *files*

●指定されたファイル(*files*)の内容に対してテストを行い、ファイルの種類を判定する。

【引数】

files：ファイル名

【オプション】(＊)

オプション	説明	対応
-c	：/etc/magic をマジックファイルとして使用する。	SUN EWS OMR PCU UXW
-f *ffile*	：*ffile* で指定したファイルからテスト対象のファイル名を得る。	HPU AIX DEC SOL UXW
-m *mfile*	：*mfile* で指定するマジックファイルを使用する。	
-h	：シンボリックリンクには従わない。	UXW

【注】

・このコマンドの結果は、完全であるとは限らないが、大方は信用できる。

【使用例】

```
%file exfile exfile.c exfile.txt⏎  ←—— exfile,exfile.c,exfile.txt の各ファイルの内容の種類を判定する
exfile:       mc68020 demand paged dynamically linked executable not stripped
exfile.c:     c program text
exfile.txt:   ascii text
%
```

exfile　　：MC68020 の実行形式プログラム
exfile.c　：C 言語のソースプログラム
exfile.txt：アスキー形式のテキストデータ

find

ファイルの存在する位置の探査と出力

find files

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

find *paths options*

●論理式（expressions）で表現されたファイルに一致するものを、指定したパス名（*paths*）以下の各ディレクトリについて捜し出す。

【引数】

paths	：ファイルを捜す対象となるディレクトリの開始位置。この下のすべてのサブディレクトリの中から検索する。

【オプション】

-name *file*	：ファイル名 (*file*) に一致するファイルが存在すれば真。エスケープ (/) を指定すれば、シェルの引数のシンタックスが使える。	
-perm *on*	：保護コードが *on* (8進数) に一致すれば真	
-type *c*	：ファイルタイプが *c* に一致すれば真	
c = b	：ブロック型特殊ファイル	
c	：文字型特殊ファイル	
d	：ディレクトリ	
f	：通常ファイル	
l	：シンボリックファイル	SUN HPU NWS AIX DEC EWS PCU UXW
p	：名前付きパイプ (FIFO)	SUN HPU AIX DEC EWS OMR UXW
s	：ソケット	SUN HPU NWS AIX DEC
-links *n*	：リンク数が *n* に一致すれば真	

-user *name*　：ファイルの所有者が *name* に一致すれば真

-nouser　：ファイルの所有者が/etc/passwd になければ真　SUN NWS AIX UXW

-group *name*：グループ名（id）が *name* に一致すれば真

-nogroup　：グループ名（id）が/etc/group になければ真　SUN NWS AIX SOL UXW

-size *n* [*c*]　：ファイルの長さが *n* ブロックであれば真
1ブロック=512バイト
n の後に c を付ければ *n* はバイト数と見なす。　SUN HPU AIX DEC PCU UXW

-inum *n*　：i ノード番号が *n* に一致すれば真　SUN HPU NWS AIX DEC EWS PCU UXW

-atime *n*　：ファイルが n 日以内にアクセスされていれば真

-mtime *n*　：ファイルが n 日以内に更新されていれば真

-ctime *n*　：ファイル属性が n 日以内に変更されていれば真

-exec *cmd*　：指定したコマンド(cmd)を実行し、終了ステータスが0であれば真

-ok *cmd*　：標準出力にコマンド（*cmd*）を出力し、これに対して y を入力する形式で実行する。この他は-exec に同じ。

-print　：常に真。捜されたファイル名のパス名を出力する。

-ls　：常に真。捜されたファイル名のパス名とそれに関連する統計情報を出力する。　SUN NWS AIX

-cpio *dev*　：常に真。捜されたファイルを cpio コマンド(5120バイトレコード）の形式で *dev* に書き出す。

-ncpio　：常に真。-cpio において、文字コードでヘッダ情報を書き込む。　EWS

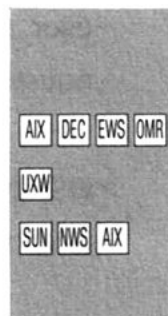
find

-newer *file*　：捜されたファイルが*file*で指定されたファイルより後に修正されていれば真

-depth　：常に真。ディレクトリ中のすべてのエントリをディレクトリより先に調べる。

-xdev　：常に真。指定したパス名のあるファイルシステムだけを探索対象とする。

AIX DEC EWS OMR UXW

SUN NWS AIX

find

【使用例】

ホームディレクトリとそのサブディレクトリにファイル名がcoreに一致するものを捜し、これをすべて表示する。

```
%find ~ -name core -print⏎
/usr/users/funamoto/c/core
/usr/users/funamoto/ctest/core
%
```

finger

ユーザの情報の出力

BSD
SUN HPU NWS AIX
DEC EWS SOL
PCU

書式

finger ［*options*］ *names*

●現在ログインしているユーザにログイン名、名前、端末名、書き込み状況、アイドルタイム、ログイン時刻、住所、電話番号を出力する。

【引数】

names：ユーザ名

【オプション】

-m ：ユーザ名だけを突き合わせる。

-l ：詳細情報の出力

-p ：.plan ファイルの内容を出力しない。

-s ：簡易情報の出力

-q ：ログイン名、端末名、ログイン時刻を出力する。 SUN AIX DEC EWS SOL

-i ：ログイン名、端末名、ログイン時刻、アイドルタイムを出力する。 SUN AIX DEC EWS SOL

-b ：詳細情報からホームディレクトリとシェルを除く。 SUN AIX DEC EWS SOL

-f ：ヘッダを出力しない。 SUN AIX DEC EWS SOL

-w ：簡易情報から名前を除く。 SUN AIX DEC SOL

-h ：.project ファイルの内容を出力しない。 SUN AIX DEC EWS SOL

【使用例】

```
%finger⏎ ←――――――――― ログインしているユーザの情報を表示する
Login       Name                TTY Idle    When    Where
root     System PRIVILEGED Ac  co    4 Tue 16:29
funamoto Susumu Funamoto       p0      Tue 16:28  cs102
%finger -l⏎ ←―――――――――― ログインしているユーザの詳細情報を表示する
Login name: root                    In real life: System PRIVILEGED Account
Directory: /                        Shell: /bin/csh
On since Oct 24 16:29:46 on console 5 minutes 2 seconds Idle Time
No unread mail
No Plan.

Login name: funamoto                    In real life: Susumu Funamoto
Directory: /usr/users/funamoto          Shell: /bin/csh
On since Oct 24 16:28:40 on ttyp0 from cs102
12 seconds Idle Time
New mail received Tue Oct 24 16:07:11 1989;
  unread since Tue Oct 24 16:07:12 1989
No Plan.
%
```

Login, Login name　：ログイン名
Name, In real life　：名前
TTY　：端末名
When　：ログイン時刻
Directory　：ホームディレクトリ
Shell　：ログインシェル
Idle　：アイドルタイム

fold

ファイル中の行の折りたたみ

BSD
HPU NWS AIX
DEC OMR SOL
PCU

書式

fold ［*option*］ *files*

●指定したファイル（*files*）の各行を一定幅に調整する。

【引数】

files ：ファイル名

【オプション】

省略時 ：行の幅を 80 とする。

-*w* ：行の幅（*w*）を指定する。

【使用例】

```
% cat exfile.dat ⏎
Funamoto Susumu.....Tokyo.....(3722)1234
Date Masamune.......Miyagi....(77)4567..
Takeda Shingen......Yamanashi.(34)8901..
% fold -20 exfile.dat ←――――――― 1 行 20 文字で折り返す
Funamoto Susumu.....
Tokyo.....(3722)1234
Date Masamune.......
Miyagi....(77)4567..
Takeda Shingen......
Yamanashi.(34)8901..
%
```

grep (egrep, fgrep)

ファイル中の指定した文字列やパターンを走査して出力

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

global regular expression printer

書式

grep [*options*] *expression* [*files*]
egrep [*options*] [*expression*] [*files*]
fgrep [*options*] [*strings*] [*files*]

grep

●指定したファイル(*files*)を検索して、指定したパターン(*expression*)を含む行を出力する。

- grep は、ex(ed)で使用できる正規表現形式を使用する。
- egrep は、grep より多くの正規表現形式を使用できる。
- fgrep は、固定長文字列だけをパターンとして指定し、高速検索する。

【引数】

files ：検索対象のファイル名
　省略時：標準入力から読み込む。
expression ：検索パターン
strings ：検索文字列

【オプション】

-b ：各行の先頭にその行があるブロック番号を入れる。
-c ：パターンに一致した行の数を出力する。
-i ：大文字と小文字を区別しない。
-l ：パターンに一致した行を含むファイル名を出力する。
-n ：各行の先頭に行番号を入れる。
-s ：エラーメッセージを出力しない。 HPU DEC EWS OMR SOL PCU UXW

エラーメッセージだけを出力する。 SUN NWS AIX
-u ：パターンを含む行を除くすべての行を出力する。 SUN PCU
-h ：ファイル名を出力しない。 SUN SOL UXW

-v　：一致しない行を出力する。

-w　：指定した正規表現を単語とみなして検索する。

SUN NWS AIX DEC SOL

-x　：完全に一致した行だけを出力する（fgrepだけ）。

SUN HPU NWS DEC EWS OMR

-e *expression*：検索パターンを指定する（パターンが-で始まっているときに有効)。egrepのときは*expression*はリテラル文字列。

SUN HPU NWS DEC EWS OMR UXW

-f *file*　：正規表現(egrep)または文字列のリスト(fgrep)がファイル(*file*)から読み込まれる。

SUN HPU NWS DEC EWS OMR UXW

-p *sep*　：一致したパターンを含むパラグラフを出力する。パラグラフは、指定したセパレータ（*sep*）で区切られたもの。

AIX

grep

<限定正規表現に使う特殊文字>

パターン	意味
.	任意の1文字に一致
¥	特殊文字の機能を取り消し、通常文字として扱う。
*	この直前の文字の0回以上の繰り返しに一致
^	行の先頭と一致
$	行の末尾と一致
[*str*]	文字列(*str*)の内の1文字に一致
[*chr1-chr2*]	2つの文字(*chr1*,*chr2*)の範囲の1文字に一致
[^*str*]	文字列(*str*)の内の文字以外の文字に一致

【使用例】

```
%cat -n exfile.c⏎
     1  #include <stdio.h>
     2  #define DEL1 '¥010'
     3  #define DEL2 '¥137'
     4  main()
     5  {
     6          int c,p;
     7          p='¥000';
     8          while((c=getchar())!=EOF){
     9                  if (c!=DEL1 && c!=DEL2){
    10                          putchar(c);
    11                  } else {
    12                          if (
    13                              (p>=129 && p<=159)
    14                              || (p>=224 && p<=252)){
    15                                  putchar(c);
    16                          }
    17                  }
    18                  p=c;
    19          }
    20          exit(0);
    21  }

%grep -n char exfile.c⏎
8:      while((c=getchar())!=EOF){
10:                     putchar(c);
15:                             putchar(c);
```

exfile.c の中に'char'という文字列を含む行を捜し、行番号を付けて表示する

行番号

```
%ls -l⏎
total 222
-rwxr-xr-x  1 funamoto    24576 Oct 30 04:52 a.out
drwxr-xr-x  3 funamoto      512 Oct 17 07:36 cc
-rw-r--r--  1 funamoto  2188090 Oct 17 03:28 core
-rwxr-xr-x  1 funamoto    24576 Oct 29 04:45 exfile
-rw-r--r--  1 funamoto      282 Oct 16 23:30 exfile.c
-rw-r--r--  1 funamoto       30 Oct 28 23:27 exfile.dat
-rw-r--r--  1 funamoto       58 Oct 29 04:15 exfile.txt
-rwxr-xr-x  1 funamoto    24576 Oct 17 03:22 exfile1
-rw-r--r--  1 funamoto      290 Oct 25 05:34 exfile1.c
-rw-r--r--  1 funamoto      989 Oct 30 00:22 exfile1.o
-rw-r--r--  1 funamoto       58 Oct 28 23:18 exfile1.txt
-rw-r--r--  1 funamoto      282 Oct 17 07:33 exfile2.c
-rw-r--r--  1 funamoto       44 Oct 28 23:24 exfile2.txt
-rwxr-xr-x  1 funamoto       67 Oct 30 00:32 infile
drwxr-xr-x  2 funamoto      512 Oct 30 02:36 link
drwxr-xr-x  2 funamoto      512 Oct 25 05:18 mk
-rw-r--r--  1 funamoto       67 Oct 20 04:57 students.dat
-rw-r--r--  1 funamoto      453 Oct 20 04:59 total.awk

%ls -l | grep -e txt⏎  ← カレントディレクトリのディレクトリの詳細情報のうち、'txt'という文字列を含む行だけを表示する
-rw-r--r--  1 funamoto       58 Oct 29 04:15 exfile.txt
-rw-r--r--  1 funamoto       58 Oct 28 23:18 exfile1.txt
-rw-r--r--  1 funamoto       44 Oct 28 23:24 exfile2.txt
%
```

grep

<fgrep コマンド>

fgrep コマンドには、

・特殊文字をパターン指定に使用しない。

・複数のパターンを記述できる。

という特徴がある。

```
% fgrep    "funamto
            susumu
            Author" exdoc.txt ⏎
```

この例では、exdoc.txt というテキストファイルの中から funamoto、susumu、Author のいずれかに一致する行を抽出する。

<egrep コマンド>

egrep コマンドには、

・複数のパターンを指定できる。

・grep で使える特殊文字に加えて

|	(or 演算)
+	(1 回以上の繰り返し)
?	(0 回または 1 回の繰り返し)

などのパターン指定ができる特徴がある。

```
% egrep "(susumu|syotaro) funamoto" exdoc.txt ⏎
```

この例では、exdoc.txt の中から

susumu funamoto

syotaro funamoto

のいずれかの文字列に一致する行を抽出する。

groups

所属するグループ名の表示

書式

groups ［*options*］［*users*］

●ユーザ自身または指定したユーザ（*users*）の属するグループ名を出力する。

【引数】

users ：ユーザ名
　　　　省略時：自分自身

【オプション】

-p　：/etc/groupだけを参照する。 HPU

-g　：/etc/logingroupだけを参照する。 HPU

-l　：/etc/passwdだけを参照する。 HPU

【使用例】

```
%groups⏎ ←———— 自分の属しているグループ名を表示する
aigroup
%
```

head

ファイルの先頭部分の出力

head of file

BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

head [*option*] [*files*]

●指定した各ファイル（*files*）の先頭部分を出力する。

【引数】

files　：ファイル名
　省略時：標準入力から読み込む。

【オプション】

-*n*　：行数の指定
　省略時　：*n*=10

【使用例】

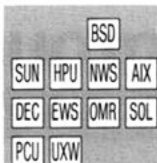
```
%head -5 exfile1.c exfile2.c⏎
==> exfile1.c <==
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥138'
main()
{

==> exfile2.c <==
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
%
```

← ファイル exfile1.c と exfile2.c の各先頭5行ずつを表示する

exfile1.c の先頭5行

exfile2.c の先頭5行

CSH

history

コマンド履歴の出力

history list

書式

history [*option*] [*n*]

●コマンド履歴を出力する。

【引数】

n　：出力するコマンドの行数。
　　　省略時：全履歴を表示する。

【オプション】

-r　：新しい順に表示する。

-h　：出力時に番号を付けない。

【入力書式】

% *event* [:*position*] [:*action*]

event　：参照の指定でヒストリリスト中の特定のコマンド行を選択する。

position　：コマンド行中の編集対象の位置を指定する。

action　：サブコマンドを使った編集内容を記述する。

evnt	意　味
!!	直前のコマンド行
!*n*	*n* 番目のコマンド行
!-*n*	*n* 番目前のコマンド行
!*str*	文字列（*str*）で始まる最新のコマンド行
!?*str*?	文字列（*str*）を含む最新のコマンド行

position	意　味
0	コマンド名
n	*n* 番目の単語
^	最初の単語
$	最後の単語
n-*m*	*n* 番目から *m* 番目までの単語
-*n*	0 番目から *n* 番目までの単語
n-	*n* 番目から最後の 1 つ前までの単語
*n**	*n* 番目から最後までの単語
*	1 番目から最後までの単語

action	意　味
h	フルパス指定のファイル名からファイル名部分を除いたもの
r	ファイル名から「.」以下の拡張子を除いた部分
t	パス指定を除いたファイル名の部分
s/*str1*/*str2*/	文字列 1（*str1*）を文字列 2（*str2*）に置換する

【使用例】

```
%rwho⏎ ←――――――― ネットワーク上のユーザを表示
tarou     sun01:ttyp0    Oct 28 13:50
sumiyo    sun03:ttyp1    Oct 28 13:38
root      vax01:console  Oct 29 04:40
susumu    sun07:ttyp0    Oct 28 15:26 :02
%ls⏎ ←――――――― カレントディレクトリのファイル名表示
a.out        exfile       exfile1       exfile2.txt   students.dat
cc           exfile.c     exfile1.c     exfile4.c     total.awk
core         exfile.dat   exfile1.txt   infile
coresearch   exfile.txt   exfile2.c     mk
%cc exfile⏎ ←――――――― C コンパイラ
%cc -o exfile exfile.c⏎ ←――――――― C コンパイラ（実行ファイルの名前指定）
%exfile < infile > outfile⏎ ←――――――― コマンドの実行
%history 6⏎ ←――――――― ヒストリリストを6行逆って表示する
    81  rwho
    82  ls
    83  cc exfile
    84  cc -o exfile exfile.c
    85  exfile < infile > outfile
    86  history 5
%!83⏎ ←――――――― 83番目のコマンドを再実行する
cc exfile.c
%!cc⏎
cc -o exfile exfile.c ←――――――― 文字列ccを含むコマンド列の最新のものを実行する
%
```

history

hostid

ホスト識別番号の出力

host identifier

書式

hostid ［*option*］

●使用中のホスト識別番号(16進)を出力する。

【オプション】

hex　：16進の番号

adr　：インターネットアドレス

name　：ホスト名

【使用例】

```
%hostid⏎ ←―――――― 使用しているホストの番号を表示する
1700c3a1
%
```

hostid

hostname

使用中のホスト名の出力

BSD
SUN HPU NWS AIX
DEC EWS OMR
PCU UXW

name of current host

書式

hostname ［*option*］

●現在使用中のホスト名を出力する。

【オプション】

-s　：ドメイン名の部分を出力しない。

AIX

【使用例】

```
%hostname⏎ ←――― 使用しているホストの名前を表示する
sun01
%
```

hostname

id

ユーザ名・ID、グループ名・IDの出力

SVR
SUN HPU AIX
DEC OMR
PCU UXW

identifier

書式

id [*options*]

●要求されているプロセスのユーザ名、ユーザID、グループ名、グループIDを出力する。

【オプション】

-g ：グループIDだけを出力する。 DEC

-n ：-uまたは-gオプションが指定されているときにIDの代わりに名称を表示する。 DEC

-u ：ユーザIDだけを出力する。 DEC

-a ：起動したプロセスのグループIDとユーザ名のすべてを表示する。 UXW

【使用例】

```
%id⏎ ←――――――自分のユーザ ID（名前）、グループ ID（名前）を表示する
uid=102(funamoto) gid=200(aigroup) groups=200(aigroup)
%
```

indent

C言語プログラムソースの清書

BSD
SUN HPU NWS AIX
DEC OMR

書式

indent [*options*] [*infile*] [*outfile*]

●指定したファイル(*infile*)に対してインデンテーションを行う。

【引数】

infile ：ファイル名

outfile ：結果出力先のファイル名

省略時：結果は infile に書き出され、元の内容は.B*infile* に保存される。

【オプション】

-bc ：宣言文のコンマの後で改行する。

-nbc ：-bc を無効にする。

-bl ：条件文（if 文）の{と}をそれぞれ改行する。

-br ：条件文（if 文）の{を同じ行におく。

-cd*n* ：宣言文のコメントのスタート位置（*n*）を設定する。

省略時：*n*=33

-c*n* ：コメントのスタート位置(*n*)を設定する。

省略時：*n*=33

-dj ：宣言文を左寄せにする。

-ndj ：宣言文を命令コードと同じ位置にする。

-d*n* ：コメントだけの行のコメントのスタート位置（*n*）を設定する。

省略時：*n*=0

-i*n* ：字下げの空白数（*n*）を設定する。

省略時：*n*=4

-l*n* ：1行の最大数（*n*）を設定する。

省略時：*n*=75

-v　：入力行で1行のものを2行に分割する場合にはメッセージを出力する。

-nv　：-v を無効にする。

【使用例】

```
% cat exprog.c ⏎
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
int c='¥000';
while((c=getchar())!=EOF){
if (c!=DEL1 && c!=DEL2){
putchar(c);
}
}
exit(0);
}
% indent -i2 exprog.c exprogout.c ⏎
% cat exprogout.c ⏎
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
  int c='¥000';
  while((c=getchar())!=EOF){
    if (c!=DEL1 && c!=DEL2){
      putchar(c);
    }
  }
  exit(0);
}
%
```

（最初のリスト）インデントされていない見づらいリスト

（indent -i2 exprog.c exprogout.c）←字下げ2字分インデントする

（後のリスト）インデント後のリスト

jobs

ジョブの状況の出力

CSH

list active jobs

書式

jobs [*option*]

●現在のジョブの状態を出力する。

【オプション】

-l　：プロセス番号も付加して出力する。

【使用例】

```
%cc exfile.c⏎ ←―――――― C言語のプログラムexfile.cをコンパイルする
^Z ←―――――――――――――――― このジョブが実行を中断する
Stopped
%jobs -l⏎ ←――――――――― ジョブの状態を表示する
[1]  +   868 Stopped      cc exfile.c ←―ジョブ番号=1の実行が中断
%bg⏎ ←―――――――――――――― このカレントジョブをバックグラウンドで実行する
[1]     cc exfile.c &
%jobs⏎ ←―――――――――――― ジョブの状態を表示する
[1]     Running           cc exfile.c ←―― このジョブが実行中である
%

[1]     Done              cc exfile.c ←―― ジョブが終了した

%
```

Running　　：実行中

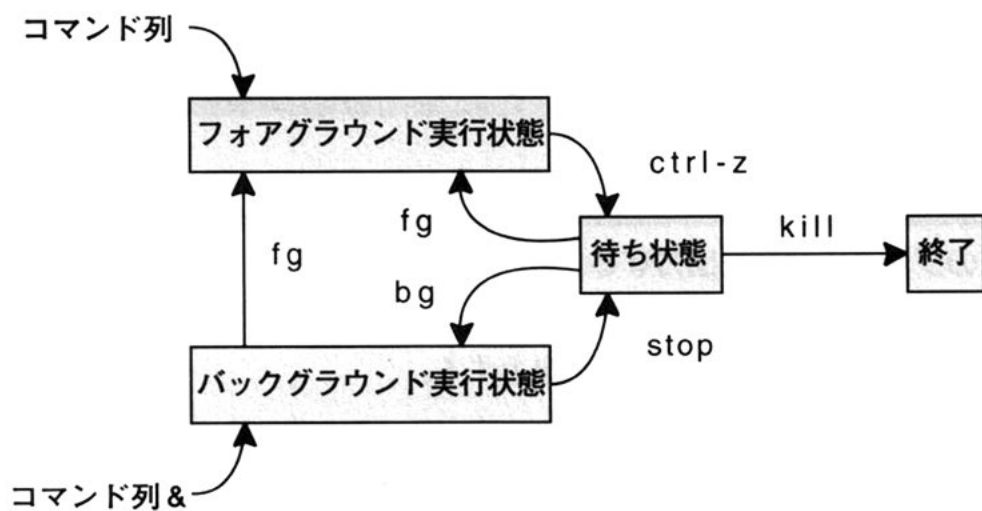
Stopped　　：一時停止

Terminated：強制終了

Done　　　：正常終了

Exit　　　 ：異常終了

●ジョブの状態遷移と制御コマンド

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

join

2つのファイルの結合

join database files

書式

join ［*options*］ *file1 file2*

●指定された2つのファイル（*file1* と *file2*）を関係づけて結合する。

【引数】

file1 ：ファイル名
　ファイル名が - であれば標準入力から読み込む。

file2 ：ファイル名

【オプション】

-a*n* ：通常の出力に加えて、対にできなかった行も出力する。*n* は1または2で、それぞれ *file1* と *file2* を指定したことになる。3を指定すれば、*file1* と *file2* の両方を対象とする。 SUN HPU AIX DEC EWS OMR SOL UXW

-e *s* ：空の出力フィールドを指定した文字列（*s*）で置き換える。

-j*n m* ：*n* 番目（*n*＝1、2）のファイルの *m* 番目のフィールドで結合する。

-o *list* ：各出力行を指定したフィールド（*list*）で構成する。

-t*c* ：指定した文字（*c*）を区切り文字とする。

-1m ：-j*1m* と同じ働きをする。 DEC

-2m ：-j*2m* と同じ働きをする。 DEC

【注】

・*file1* と *file2* は、結合されるフィールドについてASCII順にソートされていなくてはならない。

・フィールドは、空白、タブ、改行で区切られる。

【使用例】

氏名のデータファイル (name.dat) と得点のデータファイル (score.dat) を関係づけて表示する。

```
%cat name.dat↵
1 m susumu
2 f sumiyo
3 m tarou          } 番号と氏名のデータ
4 m hiroshi
5 f hanako
%cat score.dat↵
1 90 80 70
2 70 80 90
3 40 20 50         } 番号と得点のデータ
4 10  5 20
5 60 70 80
%join name.dat score.dat↵
1 m susumu 90 80 70
2 f sumiyo 70 80 90
3 m tarou 40 20 50        } 氏名と得点が関係づけられたデータの表示
4 m hiroshi 10 5 20
5 f hanako 60 70 80
%
```

kill

ジョブの強制終了

kill processes

CSH

書式

kill ［*option*］ *pids*

●指定したプロセス（*pids*）を強制終了させる。

【引数】

pids　：強制終了させたいプロセスのid

【オプション】

-signal　：プロセスに送る信号の種類の指定
　　　　省略時：TERM,15と解釈

【注】

・指定できる信号の種類はkill -lで表示できる。

・kill -9で確実に強制終了させることができる。

【使用例】

```
%cc exfile.c⏎ ←── Cのプログラム（exfile.c）をコンパイルするジョブを起動する
^Z ←──────────── このカレントジョブを中断する
Stopped
%ps⏎ ←─────────── ジョブの状態を表示する
  PID TT STAT  TIME COMMAND
  913 p0 S     0:01 -csh (csh)
  924 p0 T     0:00 cc exfile.c
  927 p0 T     0:00 as -o exfile.o -mc68020 /tmp/ccom.924.1.s
  928 p0 R     0:00 ps
%kill 924⏎ ←──── コンパイル・ジョブ（924）を強制終了する
%

[1]    Exit 1              cc exfile.c←コンパイル・ジョブが強制終了した

%ps⏎ ←─────────── ジョブの状態を表示する
  PID TT STAT  TIME COMMAND
  913 p0 S     0:01 -csh (csh) ⎫
  929 p0 R     0:00 ps         ⎭ コンパイル・ジョブは存在していない
%
```

last

最後にログインした情報の出力

indicates last logins

BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
UXW

書式

last [*options*] [*users*] [*ttys*]

●指定したユーザ（*users*）または端末（*ttys*）で、最後にログインしたユーザ名や時間などの情報を出力する。

【引数】

users　：ログイン情報を入手したいユーザ名
ここに文字列"reboot"を指定すれば、リブート間の平均時間が出力される。
省略時：全ユーザを対象とする。

ttys　：ログイン情報を入手したい端末名
省略時：全端末を対象とする。

【オプション】

-*n*　：表示する行数（*n*）　NWS

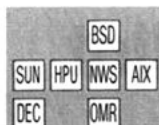
-f *file*　：情報を入手するファイル名（*file*）
省略時：/usr/adm/wtmp
/var/adm/wtmp　SUN SOL UXW

last

【使用例】

ユーザ名 funamoto のセッション情報を最近のものから 7 回分表示する。

leave
指定時刻の通知

BSD
SUN HPU NWS AIX
DEC OMR

書式

leave ［*hhmm*］

●指定した時刻（*hhmm*）およびその5分前、1分前と時刻経過後は1分ごとにメッセージを出力する。

【引数】

省略時　：時刻の指定を促すメッセージを出力する。
hh　：時間の指定（24時間表示または12時間表示）
mm　：分の指定

【使用例】

```
% leave 1630 ⏎ ←―― 16：30にアラームをセットする
Alarm set for Thu Sep 12 16:30:00 1991
% You have to leave in 5 minutes ←―― 5分前のメッセージ
Just one more minute! ←―― 1分前のメッセージ
Time to leave!←―― 設定時刻のメッセージ
You're going to be leave!←―― 1分後のメッセージ
You're going to be leave!←―― 2分後のメッセージ
```

【注】

・leaveはログオフすれば終了する。
・killコマンドを使って終了させるときには、-9オプションを付ける。

lint

C言語ソースプログラムの文法チェック

書式

lint [*options*] *files*

●C言語のソースプログラム(*files*)を読み込み、文法チェックをし、結果を標準出力へ書き出す。

【引数】

files ：C言語のソースプログラムファイル

【オプション】(＊)

-a	：long 変数への不正な値の代入を報告する。
-b	：実行されない break 文を報告する。
-c	：移植性のない cast を報告する。
-C*lib*	：llib-l*lib*.ln のライブラリを作成する。
-D*name*=*def*	：cc コマンドに同じ。
-D*name*	：cc コマンドに同じ。
-h	：バグの検出や型の訂正をする。
-n	：標準ライブラリに対して互換性のチェックをしない。
-u	：未定義の関数や変数のチェックをしない。
-v	：未使用の関数の引数をチェックしない。
-x	：外部宣言されていながら使用されていない変数を報告する。
-z	：未定義の構造体をチェックしない。

【使用例】

```
% cat -n exprog.c ⏎
     1  #include <stdio.h>
     2
     3  main(int argc,char *argv[])
     4  {
     5          int     i;
     6          int     j;
     7          float   f;
     8
     9          f=1;
    10          if (argc<2){
    11                  errproc();
    12                  exit(0);
    13          }
    14          else{
    15                  for (i=1;i<argc;i++)
    16                          printf("%x ¥n",atoz(argv[i]));
    17          }
    18  }
    19  errproc()
    20  {
    21          printf("input error");
    22          return(1);
    23          printf("erorr2 error2");
    24  }
% cc exprog.c ⏎
ccom: Warning: exprog.c, line 23: statement not reached
        printf("erorr2 error2");
      -------------------------^
ld:
Undefined:
atoz
% lint exprog.c ⏎
```

cc コマンドによるエラーメッセージ

lint

```
exprog.c                              lintコマンドによる詳細な診断メッセージ
==============
Warning: (9)  f set but not used in function main
Warning: (6)  j unused in function main
Warning: (18)  main() returns random value to invocation environment
Warning: (24)  function errproc has return(e); and return;
warning: statement not reached
    (23)



==============
name used but not defined
    atoz        exprog.c(16)
function returns value which is always ignored
    errproc
%
```

ln

ファイルのハードまたはシンボリックリンク

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

link files

書式

❶ ln [*options*] *file1* [*file2*]
❷ ln [*options*] *files dir*

❶指定したファイル(*file1*)を別のファイル(*file2*)にリンクする。
❷指定したファイル(*files*)をディレクトリ(*dir*)にリンクする。

【引数】

file1 ：リンクを生成するファイル名
file2 ：リンクを生成する時にターゲットとなるファイル名
files ：リンクを生成するファイル名
dir ：ターゲットとなるディレクトリ名

【オプション】

-f ：ハードリンクを生成する（スーパーユーザのみ）。
強制的にリンクを生成する。 SUN HPU AIX DEC EWS OMR SOL UXW

-s ：シンボリックリンクを生成する。

-i ：存在するファイルの時には問い合わせをする。 DEC

【使用例】

```
%ls -l⏎
total 24
-rwxr-xr-x  1 funamoto    24576 Oct 30  1989 exfile
%ln exfile ex1⏎ ←──── ex1 というファイル名で exfile にハードリンクする
%ls -l⏎ ←──── リンク数が増加している
total 48
-rwxr-xr-x  2 funamoto    24576 Oct 30  1989 ex1
-rwxr-xr-x  2 funamoto    24576 Oct 30  1989 exfile
%ln -s exfile ex2⏎ ←──── ex2 というファイル名で exfile にソフトリンクする
%ls -l⏎
total 49
-rwxr-xr-x  2 funamoto    24576 Oct 30  1989 ex1
lrwxrwxrwx  1 funamoto        6 Oct 30  1989 ex2 -> exfile
-rwxr-xr-x  2 funamoto    24576 Oct 30  1989 exfile
%
```

lock

端末のロック

書式

lock [*option*]

●パスワードが入力されるまで端末をロックする。

【オプション】

-*n*　：タイムリミット(n：分)を指定する。
　　省略時：n=15

lock

login

ログイン（セッション開始）

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

login [*options*] [*name*]　SUN NWS DEC
login [*name* [*env_vals*]]　HPU EWS PCU
login [*options*] [*name* [*env_vals*]]　AIX OMR

login

●指定したユーザ名（*name*）でログインする。

【引数】

name　：ユーザ名
省略時：システムからユーザ名とパスワードの入力要求がされる。

env_vals：環境変数の指定　AIX EWS OMR CPU UXW
ただし、PATH と SHELL の変更はできない。

【オプション】

省略時：以前に設定された環境変数の値は無効にされる。
-p　：現在の環境変数の値を保存する。
-r　：リモートホストにログインする。　DEC

【使用例】

```
%login⏎ ←――――――――――― 別のセッションを開始する
login: funamoto⏎ ←―――――――― ユーザ名の入力
Password:______⏎ ←―――――――― パスワードの入力
Last login: Sun Oct 29 13:36:54 on ttyp0←―― 前回のログイン情報
SunOS Release 4.0_Export (GENERIC) #1: Tue Apr 25 16:55:29 JST 1989
You have mail.←――――――――――― メールが着信している
Sun Oct 29 13:38:06 JST 1989 ←―――― 現在の時刻
%
```

logname

ログイン名の出力

login name

書式

logname

●ユーザがシステムにログインした時のログイン名を出力する。
すなわち、環境変数 LOGNAME の内容である。

【使用例】

```
%logname⏎ ←―――――――――― 現在のログイン名を表示する
funamoto
%echo $LOGNAME⏎ ←―――――― 環境変数 LOGNAME の内容を表示する
funamoto ←―――――――――――― 両者は同一
%
```

logout

ログアウト（セッション終了）

CSH

logout

書式

logout

●ログアウト（セッションの終了）をする。

【使用例】

```
%logout⏎ ←―――― セッションを終了する

login: ←――――― 次のセッション開始を求めるメッセージが表示される
```

lp

ラインプリンタへの出力要求

line printer

書式

lp ［*options*］［*files*］

SVR HPU AIX DEC EWS OMR SOL PCU UXW

●指定したファイル(*files*)とその関連情報をラインプリンタに出力する準備をする。

【引数】

files　：出力したいファイル名
　省略時 または －　：標準入力から読み込む。

【オプション】

-c　：ファイルのコピーを作成し、このコピーの内容を出力する。

-d*dest*　：出力するプリンタまたは、プリンタ・クラス(*dest*)を指定する。

-i　：印字時に字下げをしない。　OMR

-m　：ファイルの出力終了後にメールを送る。　HPU AIX OMR SOL EWS PCU UXW

-n*num*　：コピーの部数（*num*）を指定する。
　省略時：1部

-o*option*：プリンタまたは、プリンタ・クラスに特有のオプション（*option*）を指定する。　HPU AIX OMR SOL EWS PCU UXW

-s　：メッセージを出力しない。　HPU EWS OMR SOL PCU UXW

-t*title*　：ヘッダ・ページにタイトル（*title*）を出力する。　HPU AIX OMR SOL EWS PCU UXW

-w　：ファイルの出力終了後に端末にメッセージを送る。ログインされていなければメールを送る。　HPU AIX OMR SOL EWS PCU UXW

【注】

・出力要求を取り消すには cancel コマンドを使用する。

・プリンタの状況を知るには lpstat コマンドを使用する。

【使用例】

```
%lp exfile.c⏎ ←――――― ファイル exfile.c をプリンタに出力要求する
request id is LP01-2 (1 file)
%
```

lpq

スプールにあるプリントジョブの状況の出力

line printer queue

BSD
SUN NWS AIX
DEC EWS SOL
PCU UXW

書式

lpq [*options*] [*jobs*] [*users*]

●指定したジョブ番号（*jobs*）または、ユーザ名（*users*）のスプールキューの状況を出力する。

【引数】

jobs　：出力するプリントジョブを発行したジョブ番号
users　：出力するプリントジョブを発行したユーザ名
jobs、*users*：省略時：すべてのプリントジョブが対象となる。

【オプション】(＊)

+　：定期的に空になるまでスプールキューの状態を報告する。
+*n*　：*n* 秒間 lpq が停止し、この間にスプールキューの状況を調べる。
-l　：詳細情報を表示する。
-P*ptr*　：指定のプリンタ名（*ptr*）を指定する。
省略時：デフォルトプリンタ、または環境変数 PRINTER で指定されているプリンタ。

【使用例】

・lpr コマンドを参照。

lpr

プリンタ(スプール)に対してのジョブの出力

BSD SUN NWS AIX DEC EWS SOL PCU UXW

line printer

書式

lpr [*options*] [*files*]

●ラインプリンタのスプーラを使って、プリンタの空いている時に指定したファイル（*files*）の内容を出力する。

【引数】

files　：印刷するファイル名

【オプション】（＊）

-# *n*　：コピー枚数（*n*）の指定

-P*ptr*　：特定のプリンタ名（*ptr*）の指定

-C*name*　：バーストページにシステム名の代わりの文字列（*name*）を出力する。

-J*name*　：バーストページにジョブ名の代わりの文字列(*name*)を出力する。

-T*title*　：ファイル名の代わりに、指定したタイトル(*title*)を出力する。

-i*n*　：出力をインデントする。各行の先頭に *n* 個の空白を入れる。

-*nfont*　：フォント位置(*n*)にマウントするフォント(*font*)を指定する。　SUN NWS DEC SOL

-w*n*　：1ページの行数（*n*）の指定。

-r　：スプール完了後または印刷完了後に、ファイルを削除する。

-m　：完了後にメールを送信する。

-h　：バーストページを印刷しない。

-s　：スプールのディレクトリへのシンボリック・リンクを使用する。

＜フィルタ・オプション＞

- -p　：pr コマンドと同一の形式で出力する。
- -l　：制御文字を印刷し、改行を無視する。
- -t　：troff コマンドの出力データが含まれている。
- -n　：ditroff コマンドの出力データが含まれている。
- -d　：tex コマンドの出力データが含まれている。
- -g　：plot コマンドの出力データが含まれている。
- -v　：ラスターイメージが含まれている。
- -c　：cifplot コマンドの出力データが含まれている。
- -f　：各行の先頭文字を FORTRAN のキャリッジ制御文字と見なす。

【使用例】

```
%lpr -Plp exfile.c⏎
%lpq⏎
lp is ready and printing
Rank   Owner      Job  Files                    Total Size
active funamoto   257  exfile.c                 282 bytes
%lprm 257⏎
dfA257sun07 dequeued
cfA257sun07 dequeued
%lpq⏎
no entries
%
```

lprm

スプールにあるプリントジョブの削除

BSD
SUN NWS AIX
DEC EWS SOL
PCU UXW

line printer remove

書式

lprm ［*options*］［*jobs*］［*users*］

●ラインプリンタのスプールキューから指定したジョブ(*jobs*)を削除する。

【引数】

jobs　：削除するジョブ番号
(lpq -l で表示される番号)

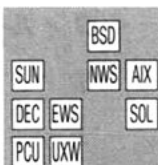
users　：該当するユーザの発行したジョブを削除
(スーパーユーザのみ)

【オプション】

-　：ユーザの全ジョブを削除する。

-P*ptr*　：特定のプリンタキュー（*ptr*）の指定
省略時：デフォルトプリンタまたは、環境変数
PRINTER で指定したプリンタ。

【使用例】

・lpr コマンド（p.240）を参照。

lpstat

プリンタの状況の出力

line printer status

SVR HPU AIX DEC EWS OMR SOL PCU UXW

書式

lpstat ［*options*］

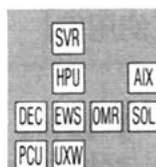
●ラインプリンタのスプーリング情報を出力する。

【オプション】

省略時	：すべての情報を出力する。
-a ［*list*］	：プリンタやプリンタクラス（*list*）の受理状態を出力する。
-c ［*list*］	：クラス名（*list*）とメンバを出力する。
-d	：プリンタ指定をしない場合の出力プリンタ名を出力する。
-o ［*list*］	：出力リクエストの状態を出力する。*list* はプリンタ名、クラス名、リクエストのリスト。
-p ［*list*］	：プリンタ（*list*）の状態を出力する。
-r	：スケジューラの状態を出力する。
-s	：状態の一覧を出力する。スケジューラ、出力プリンタ名、クラス名、メンバーリスト、装置リスト。
-t	：すべての状態情報を出力する。
-u ［*list*］	：ユーザ（*list*：ログイン名）の出力リクエストの状態を出力する。
-v ［*list*］	：プリンタ名（*list*）とそれに対応する装置のパス名を出力する。

【注】

・プリンタへの出力要求は、lp コマンドを使う。

・出力要求の取り消しは、cancel コマンドを使う。

【使用例】

```
%lpstat -t⏎ ←――――――――――― プリンタのスプール情報を表示する
scheduler is running ←――――――――――― スケジューラが動作中
system default destination: LP01←―― プリンタ名LP01
device for LP01: /dev/lp
LP01 accepting requests since Dec 13 13:53
printer LP01 is idle.  enabled since Dec 13 13:54
%
```

ls

ディレクトリの内容の出力

list directory

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

ls [*option**s***] [*name**s***]

●指定されたファイルまたはディレクトリ(*name**s***)に関する情報を出力する。複数のファイル、ディレクトリの指定をすればファイル、ディレクトリの順に処理する。

【引数】

files ：ファイル名またはディレクトリ名の指定
省略時：現在のディレクトリが指定されたものとみなす。

【オプション】

省略時：ファイル名またはディレクトリ名だけを出力する。

-a ：すべてのエントリを出力する。ファイル名が (.) ではじまるエントリも含む。

-A ：他のオプションがないときに (.) (..) ではじまるファイル名に関して出力されない他は -a と同じ。 SUN HPU NWS AIX SOL

-b ：表示できない文字を/ddd (8 進数) で出力する。 SUN HPU AIX SOL UXW

-c ：i ノードの最終変更時刻をソートや出力に使用する。 HPU AIX DEC SOL PCU

ファイルの最終修正または最終モード変更時刻をソートや出力に使用する。 SUN NWS SOL DEC UXW

-C ：マルチカラム形式で、エントリを縦方向にソートして出力する。

-d ：引数がディレクトリの時に、名前だけを出力する。-l と一緒に使用すれば、ディレクトリの状態を知ることができる。

ls

-f ：引数をディレクトリと見なし、各スロットにあるディレクトリ名を出力する。

-F ：ディレクトリに対して、名前の後に/を付け、実行可能なファイルに対して＊を付けて出力する。

-g ：-lの出力から所有者名を除いて出力する。

-i ：各ファイルに対してi番号を最初のカラムに追加して出力する。

-l ：ロング形式で出力する。ファイルのモード、リンク数、所有者名、最終修正時刻が付加される。
特殊ファイルの時には、サイズフィールドに装置のメジャー番号とマイナー番号が入る。
シンボリックリンクされている場合には、リンクされたファイルのパスネームが->付きで出力される。 SUN NWS EWS OMR UXW

-L ：引数の名前がシンボリックされている場合には、リンク先の名前が出力される。 SUN NWS EWS

-m ：ストリーム形式で出力する。ファイル名はピリオド(.)で区切って出力する。

-n ：-lの出力で所有者名をユーザID(UID)、グループ名をグループID（GID）にして出力する。

-o ：-lの出力からグループ名を除いて出力する。

-p ：ディレクトリの場合、名前の後に/を付けて出力する。

-q ：名前の文字列の中に表示不可能な文字がある場合、これを？に置き換えて出力する。

-r ：アルファベットの逆順または修正時刻の古い順にソートして出力する。

-R ：サブディレクトリを再帰的に出力する。

-s ：各エントリに対して、間接ブロックも含めて、その大きさを出力する。

ブロック数 EWS OMR PCU

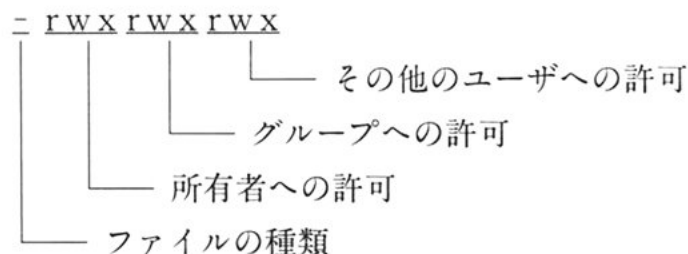
キロバイト数 SUN NWS AIX SOL DEC UXW

-t　：最終修正時刻に従って最近のものから順にソートして出力する。

-u　：-tによるソートや-lによる出力で最終修正時刻の代わりに最終アクセス時刻を出力する。

-x　：マルチカラム形式で、横方向にソートして出力する。 HPU NWS AIX SOL UXW

-1　：他のオプションがない時に、1行に1エントリの形式で出力する。 SUN HPU NWS AIX DEC UXW

<モードの形式>

<-l オプションで出力されるモード>

先頭の1文字：ファイルの種類を表す。

- ：通常ファイル

d ：ディレクトリ

b ：ブロック型特殊ファイル

c ：キャラクタ型特殊ファイル

p ：名前付きパイプ（FIFO型特殊ファイル）

l ：シンボリックリンク SUN HPU NWS AIX DEC

s ：AF_UNIXアドレス・ファミリ・ソケット SUN HPU AIX DEC UXW

l ：アクセス中に必須ロック発生 OMR

w ：ウィンドウ型スペシャルファイル EWS

<2文字目から10文字：読み書き実行の可能性>

r ：読み取り可能

w ：書き込み可能

x ：実行可能

ls

－ ：各許可の否定

【使用例】

```
                  ┌─ カレントディレクトリのディレクトリ情報を表示する。ファイル名が
%ls -a⏎ ←────┘    '.'で始まるものも対象とする
.          core        exfile.txt    exfile1.txt   link
..         exfile      exfile1       exfile2.c     mk
a.out      exfile.c    exfile1.c     exfile2.txt   students.dat
cc         exfile.dat  exfile1.o     infile        total.awk
 ('.'は親のディレクトリ、'..'は自分自身を示す)


%ls -l⏎ ←─── カレントディレクトリの詳細ディレクトリ情報を表示する
total 222
-rwxr-xr-x  1 funamoto   24576 Oct 30  1989 a.out
drwxr-xr-x  3 funamoto     512 Oct 17 07:36 cc
-rw-r--r--  1 funamoto 2188090 Oct 17 03:28 core
-rwxr-xr-x  1 funamoto   24576 Oct 29 04:45 exfile
-rw-r--r--  1 funamoto     282 Oct 16 23:30 exfile.c
-rw-r--r--  1 funamoto      30 Oct 28 23:27 exfile.dat
-rw-r--r--  1 funamoto      58 Oct 29 04:15 exfile.txt
-rwxr-xr-x  1 funamoto   24576 Oct 17 03:22 exfile1
-rw-r--r--  1 funamoto     290 Oct 25 05:34 exfile1.c
-rw-r--r--  1 funamoto     989 Oct 30  1989 exfile1.o
-rw-r--r--  1 funamoto      58 Oct 28 23:18 exfile1.txt
-rw-r--r--  1 funamoto     282 Oct 17 07:33 exfile2.c
-rw-r--r--  1 funamoto      44 Oct 28 23:24 exfile2.txt
-rwxr-xr-x  1 funamoto      67 Oct 30  1989 infile
drwxr-xr-x  2 funamoto     512 Oct 30  1989 link
drwxr-xr-x  2 funamoto     512 Oct 25 05:18 mk
-rw-r--r--  1 funamoto      67 Oct 20 04:57 students.dat
-rw-r--r--  1 funamoto     453 Oct 20 04:59 total.awk
     ↑       ↑    ↑           ↑        ↑          ↑
   モード リンク数 所有者名     容量     日付時刻   ファイル名
%ls -l exfile.*⏎ ←─── exfile.*に該当するファイルのディレクトリ情報を表示する
-rw-r--r--  1 funamoto     282 Oct 16 23:30 exfile.c
-rw-r--r--  1 funamoto      30 Oct 28 23:27 exfile.dat
-rw-r--r--  1 funamoto      58 Oct 29 04:15 exfile.txt
%
```

mail

電子メールの発信と受信

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

send and read mail

書式

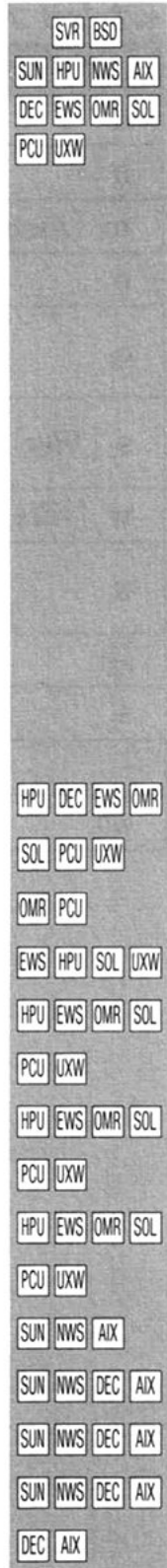
mail ［*options*］［*users*］

●指定したユーザ（*users*）にメールを発信する。受信したメールを出力する。

【書式】

users ：発信先のユーザ名
　省略時：受信メールを出力する。

【オプション】

オプション	説明	対応
-f *file*	：指定したファイル（*file*）からメールを読み出す。 省略時：メールファイルから読み出す。	
-e	：メールの出力はしなく、メールがある時に 0、ないときに 1 を出力する。	HPU DEC EWS OMR SOL PCU UXW
-p	：全メッセージを連続して出力する。 省略時：メッセージ単位にコマンドを促す。	OMR PCU
-q	：割り込みシグナルで mail コマンドを終了する。	EWS HPU SOL UXW
-r	：メッセージを着順に出力する。 省略時：最近のメッセージから出力する。	HPU EWS OMR SOL PCU UXW
-t	：全メッセージの先頭に宛名（*users*）を付ける。	HPU EWS OMR SOL PCU UXW
-v	：メールの配達についての詳細情報を表示する。	SUN NWS AIX
-i	：tty 割り込みを無視する。	SUN NWS DEC AIX
-n	：Mail.rc の読み込みを禁止する。	SUN NWS DEC AIX
-s *subject*	：メールにタイトル（*subject*）を付ける。	SUN NWS DEC AIX
-u *user*	：mail -f /user/spool/mail/user の意味。	DEC AIX

＜主なサブコマンド＞

サブコマンド	機　能
d	現在選択されているメールを削除する。
h	メールの一覧を表示する。
m ［*users*］	指定したユーザ（*users*）にメッセージを送る。
p	現在選択されているメールを表示する。
q	mail コマンドを終了する。 読まれたメールは mbox という名前のファイルに保存される。
s ［*files*］	指定したファイル（*files*）にメールを保存する。 ファイル名を省略したときは mbox に保存される。
w ［*files*］	指定したファイル（*files*）にヘッダ部分以外を保存する。
x	mail コマンドを終了する。 mbox には保存されない。
⏎	現在選択されているメールを表示する。
+	現在選択されているメールを表示する。
-	1 つ前のメールを表示する。
no	指定した番号（*no*）のメールを表示する。

【使用例】

```
%mail funamoto⏎ ←―――――――― ユーザ funamoto にメールを送信する
Subject: MEETING⏎ ←―――――――― 題目
I want to meet you.⏎ ←――――― メッセージ１行目
Please tell me your schedule.⏎ ← メッセージ２行目
.  ←―――――――――――――――― メッセージ終了
EOT
%


SunOS UNIX (sun01)

login: funamoto⏎
Password: ______⏎
Last login: Sun Oct 29 15:19:50 from cs102
SunOS Release 4.0_Export (GENERIC) #1: Tue Apr 25 16:55:29 JST 1989
You have mail. ←―――――――――― メールを受信している
Sun Oct 29 15:22:15 JST 1989
% mail⏎ ←――――――――――――― メールを読む
Mail version SMI 4.0 Sat Apr  9 01:54:23 PDT 1988  Type ? for help.
"/usr/spool/mail/funamoto": 1 message 1 new ←― 1件のメールがある
>N  1 ueno            Sun Oct 29 15:21   13/310   MEETING ←ユーザ ueno
&                                                    からのメールを受信
Message  1:
From ueno Sun Oct 29 15:21:47 1989
Return-Path: <ueno>
Received: by sun07. (4.0/SMI-4.0)
        id AA00743; Sun, 29 Oct 89 15:21:46 JST
Date: Sun, 29 Oct 89 15:21:46 JST
From: ueno (=.K¥)
Message-Id: <8910290621.AA00743@sun07.>
To: funamoto
Subject: MEETING
Status: R

I want to meet you. ←―――――― 受信メッセージ１行目
Please tell me your schedule.← 受信メッセージ２行目

& d⏎ ←――――――――――――――― メッセージを削除する
&
At EOF ←―――――――――――――― 他にメッセージはない
& q  ←――――――――――――――― mail コマンドを終了する
%
```

mail

make

プログラムやファイルの保守・更新を関連づけて行う

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

make [*options*] [*files*]

● makefile または Makefile で定義されたファイルで更新されたファイルだけのコンパイルとリンクを行う。

【引数】

files ：ターゲットファイル

【オプション】

-f *makefile* ：指定したファイル(*makefile*)をメイクファイルとするファイル名が-の時には、標準入力から読み込む。

-d ：ターゲットを再作成する理由を出力する。 SUN NWS AIX DEC EWS SOL

-e ：環境変数の内容がメイクファイル内で優先して使われる。 SUN PCU

-i ：実行したコマンドから返却されたエラーコードを無視する。

-k ：実行したコマンドが異常終了した時に、これに関係した部分のこれ以降のコマンドは実行しない。他の分岐部分のコマンドは継続して実行する。

-n ：コマンドの実行は行わずに、トレースと出力のみ行う。

-p ：マクロ定義およびターゲットの記述をすべて出力する。 SUN PCU

-q ：ターゲットファイルが最新である時に0、そうでない時に1を返却する。 SUN EWS OMR AIX DEC UXW

-r ：省略時のファイルを読み込まない。

-s ：実行前にコマンド行を出力しない。

-t ：コマンドを実行せずに、ターゲットファイルの日付を

更新する。

-dd ：依存ファイルのチェックや処理を詳細に出力する。 SUN SOL

-b ：バージョンが古いメイクファイルとの互換。 EWS OMR DEC

-m ：メモリマップを出力する。 DEC

-E ：環境の読み込みを禁止する。 NWS

-D ：読み込まれたメイクファイルのテキストを出力する。 SUN SOL

-DD：すべてのファイルのテキストを出力する。 SUN SOL

-P ：マクロ定義とターゲットの記述をすべて出力する。 SUN SOL

-S ：-k オプションの効果を取り消す。 SUN SOL DEC

<makefile ファイルの基本的書式>

target : *target1*, *target2*, ･･･, *targetn*

〈tab〉 *cmd1*

〈tab〉 *cmd2*

⋮

target ：作成対象のファイル。
この記述に続けて、この *target* を作成するために必要な一連のファイル (*target1*～*n*) を：に続けて記述する。

【使用例】

```
%ls -l⏎
total 3
-rw-r--r--  1 funamoto       37 Oct 25  1989 Makefile
-rw-r--r--  1 funamoto      290 Oct 25  1989 main.c
-rw-r--r--  1 funamoto       39 Oct 25  1989 sub.c
%cat Makefile⏎ ←――――――――― Makefileの内容を表示する。
a.out: main.o sub.o
        cc main.o sub.o
%make⏎ ←――――――――――――― Makefileを実行する
cc   -sun3 -c  main.c ←――――― main.cをコンパイルする
cc   -sun3 -c  sub.c ←―――――― sub.cをコンパイルする
cc main.o sub.o ←―――――――― オブジェクトをリンクする
%ls -l⏎
total 29
-rw-r--r--  1 funamoto       37 Oct 25 05:00 Makefile
-rwxr-xr-x  1 funamoto    24576 Oct 25 05:18 a.out
-rw-r--r--  1 funamoto      290 Oct 25 05:18 main.c
-rw-r--r--  1 funamoto      989 Oct 25 05:18 main.o
-rw-r--r--  1 funamoto       39 Oct 25 04:04 sub.c
-rw-r--r--  1 funamoto      137 Oct 25 05:11 sub.o
%ed main.c⏎ ←――――――――――― ソースプログラムを編集する
290
3p⏎
#define DEL2 '¥137'⏎
s/137/138/g⏎
.⏎
#define DEL2 '¥138'⏎
w⏎
290
q⏎
%make⏎ ←――――――――――――― Makefileを実行する
cc   -sun3 -c  main.c ←――――― 更新されたmain.cだけがコンパイルされて
cc main.o sub.o ←―――――――― 実行ファイルが作成される
%make⏎
`a.out' is up to date. ←―――― 何も更新されていなければコンパイルおよび
%
```

man

オンライン・リファレンス・マニュアル

online manual

BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
UXW

書式

❶ man [*options*] [*section*] *cmds*

❷ man [*option*] -k *keywords* SUN NWS AIX DEC UXW

❸ man [*option*] -f *files* SUN NWS AIX DEC UXW

---

❶指定したコマンド(*cmds*)のオンラインマニュアルを出力する。

❷指定したキーワード（*keywords*）で、目的とするマニュアルの先頭行を出力する。

❸指定したファイル（*files*）の表題部を出力する。

【引数】

cmds　：コマンド名

section　：コマンド・マニュアルのあるセクション番号

files　：コマンドファイル名

keywords：検索キーワード

【オプション】

- \-　：出力をパイプに渡す。 SUN NWS AIX SOL
- -t　：troff 形式にして出力する。 SUN NWS AIX SOL UXW
- -a　：/usr/man だけをマニュアルファイルとする。 NWS SOL
- -M*path*：マニュアルページを捜すパス（*path*）を設定する。 SUN AIX SOL UXW
- -P*path*：指定したパス（path）のマニュアルを参照する。 DEC
- -s　：不要な空白を削除する。 DEC
- -w　：記載事項のパス名のみ表示する。 OMR EWS
- -d　：/usr/catman の代わりにワーキングディレクトリを OMR EWS

man

対象とする。

-c　　：cal を呼び出す。

-T*term*：端末の形式（*term*）に合わせた表示をする。

<マニュアルの各項目の意味>

NAME	コマンド名と簡単な説明
SYNOPSIS	書式
DESCRIPTION	詳細説明（引数、オプションなどの説明）
FILES	関連ファイル名
SEE ALSO	関連事項
DIAGNOSTIC	メッセージ、終了ステータスの説明
BUGS	バグ、注意事項の説明

【使用例】

```
%man -k editor⏎  ←── editorというキーワードを指定してマニュアルを検索する
COFF (5)                    - common assembler and link editor output
a.out (5)                   - assembler and link editor output format
ed (1)                      - basic line editor
ex, edit, e (1)             - line editor
fontedit (1)                - a vfont screen-font editor
ld, ld.so (1)               - link editor, dynamic link editor
ldconfig (8)                - link-editor configuration
link (5)                    - link editor interfaces
sed (1V)                    - stream editor
textedit (1)                - SunView window- and mouse-based text editor
vi, view (1)                - visual display editor based on ex(1)
%   ↑                          ↑
  コマンド名                   解説
%man lprm⏎  ←── lprmコマンドのマニュアルを参照する

LPRM(1)                   USER COMMANDS                   LPRM(1)

NAME
     lprm - remove jobs from the printer queue

SYNOPSIS
     lprm [ -Pprinter ] [ - ] [  job# ...  ] [  username ...  ]
```

```
DESCRIPTION
     lprm removes a job or jobs from a printer's spooling  queue.
     Since  the  spool  directory  is protected from users, using
     lprm is normally the only method by which a user can  remove
     a job.

     Without  any  arguments,  lprm  deletes  the  job  that   is
     currently  active,  provided  that the user who invoked lprm
     owns that job.

     When the super-user specifies a  username  lprm  removes  all
     jobs belonging to that user.

 --More--(21%) ←──────── スペースキーの入力
     You can remove a specific job by supplying its job number as
     an  argument,  which you can obtain using lpq(1).  For exam-
     ple:

          example% lpq -Phost
          host is ready and printing
          Rank    Owner  Job  Files                          Total Size
          active  wendy  385  standard input                 35501 bytes
          example% lprm -Phost 385

     lprm reports the names of  any  files  it  removes,  and  is
     silent if there are no applicable jobs to remove.

     lprm kills the active printer daemon, if  necessary,  before
     removing spooled jobs; it restarts the daemon when through.

OPTIONS
     -P printer
          Specify the queue associated with a  specific  printer.
          Otherwise  the  value  of  the  PRINTER variable in the
          environment is used.  If this variable  is  unset,  the
          queue for the default printer is used.
 --More--(48%) ←──────────── スペースキーの入力

     -    Remove all jobs  owned  by  you.   If  invoked  by  the
          super-user,  all  jobs  in the spool are removed.  (Job
          ownership is determined by the user's  login  name  and
          host  name  on  the  machine  where the lpr command was
          invoked).
```

```
FILES
     /etc/printcap       printer characteristics file
     /var/spool/*        spooling directories
     /var/spool/l*/lock  lock file used to obtain the pid of  the

Sun Release 4.0   Last change: 9 September 1987                1

LPRM(1)                 USER COMMANDS                   LPRM(1)

                         current daemon and the job number of the
                         currently active job

SEE ALSO
     lpr(1), lpq(1), lpd(8)

 --More--(71%) ←──────────────── スペースキーの入力
DIAGNOSTICS
     lprm:  printer : cannot restart printer daemon
          The connection to lpd  on  the  local  machine  failed.
          This  usually  means the printer server started at boot
          time has died or is hung.  If it is  hung,  the  master
          lpd(8)  daemon  may  have  to  be  killed and a new one
          started.

BUGS
     Since race conditions are possible when  updating  the  lock
     file,  an active job may be incorrectly identified for remo-
     val by an lprm command issued with no arguments.  During the
     interval  between  an  lpq(1)  command  and the execution of
     lprm, the next job in line may have become active; that  job
     may  be  removed  unintentionally if it is owned by you.  To
     avoid this, supply lprm with the job number to remove when a
     critical job that you own is next in line.

Sun Release 4.0   Last change: 9 September 1987                2

%
```

mesg

メッセージ受け付けの可否の設定

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

message

書式

mesg ［*option*］

● write コマンドによる他ユーザからのメッセージの受け付け、または拒否を指定する。

【オプション】

省略時：現在の状態を出力する。

n　　：write コマンドによるメッセージ受信を禁止する。

y　　：メッセージ受信の許可をする。

【使用例】

mesg

mkdir

ディレクトリの作成

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

make a directory

書式

mkdir ［*option*］ *dirs*

●指定されたディレクトリ（*dirs*）を作成する。標準エントリ、すなわちディレクトリ自身を示す「.」と親ディレクトリを示す「..」が自動的に作成される。

【引数】

dirs：新規に作成するディレクトリ名

【オプション】

-p　：親ディクトリがない場合には必要に応じて作成する。　SUN SOL DEC UXW

-m　：モードを指定する。　UXW

【注】

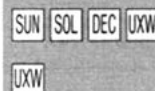
・mkdir コマンドの使用には、親ディレクトリでの書き込み許可が必要。

【使用例】

```
%ls -l⏎
total 4
drwxr-xr-x  2 funamoto     512 Oct 28 23:38 c
drwxr-xr-x  2 funamoto     512 Oct 29 02:44 csh
drwxr-xr-x  2 funamoto     512 Oct 28 23:34 lisp
drwxr-xr-x  2 funamoto     512 Oct 29 04:53 work
%mkdir exdir1 exdir2⏎  ←—— カレントディレクトリに新たなサブディレクトリ
                            exdir1 と exdir2 を作成する
%ls -l⏎
total 6⏎
drwxr-xr-x  2 funamoto     512 Oct 28 23:38 c
drwxr-xr-x  2 funamoto     512 Oct 29 02:44 csh
drwxr-xr-x  2 funamoto     512 Oct 30  1989 exdir1 } 新たに作成された
drwxr-xr-x  2 funamoto     512 Oct 30  1989 exdir2 } ディレクトリ
drwxr-xr-x  2 funamoto     512 Oct 28 23:34 lisp
drwxr-xr-x  2 funamoto     512 Oct 29 04:53 work
%
```

more(page)

テキストファイルの1ページ毎の出力

BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

page [*options*] [*files*]
more [*options*] [*files*]

●指定したファイル(*files*)または標準入力の内容を1画面分だけ出力する。

【引数】

files：出力するファイル名
　省略時：標準入力から読み込む。

【オプション】

-c　：画面スクロールをせずに、1行目から上書きして出力する。
-d　：各画面の末尾で、次に行うべきメッセージを出力する。
-f　：画面上の行ではなく、論理的な行数でカウントする。
-l　：用紙送り（CTRL＋L）を無視する。
-s　：連続する複数の空白行を1行の空白行として出力する。
-u　：アンダーラインや強調文字のエスケープコードを無視する。
+*n*　：指定した行番号（*n*）から出力を開始する。
+/*pattern*　：指定したパターン（*pattern*）を含む行の2行前から出力を開始する。
-*n*　：画面に出力する行数（*n*）を指定する。
-r　：制御文字を表示する。 UXW
-w　：出力終了の前に入力を待つ。 UXW

【使用例】

ディレクトリの詳細情報を１画面単位で表示する。表示が中断するときには次に行うべき処理に対する表示も行う。

```
%ls -al | more -d⏎
total 260
drwxr-xr-x  5 funamoto     1024 Feb  9  1990 .
drwxr-xr-x  5 funamoto      512 Nov  1 23:10 ..
-rwxr-xr--  1 funamoto      507 Feb  9  1990 .cshrc
-rw-r--r--  1 funamoto      554 Feb  9  1990 .emacs
-rw-r--r--  1 funamoto       38 Feb  9  1990 .emacs_102
-rwxr-xr-x  1 funamoto      436 Feb  9  1990 .history
-rwxr-xr--  1 funamoto      462 Feb  9  1990 .login
-rwxr-xr--  1 funamoto      127 Feb  9  1990 .profile
-rw-r--r--  1 funamoto        7 Feb  9  1990 .rhosts
-rwxr-xr-x  1 funamoto    24576 Jan 12 06:06 a.out
drwxr-xr-x  4 funamoto      512 Nov 14 07:09 cc
-rw-r--r--  1 funamoto  2188090 Oct 17 03:28 core
-rwxr-xr-x  1 funamoto    24576 Oct 29 04:45 exfile
-rw-r--r--  1 funamoto      296 Nov  2 03:04 exfile.c
-rw-r--r--  1 funamoto       30 Oct 28 23:27 exfile.dat
-rw-r--r--  1 funamoto       58 Oct 29 04:15 exfile.txt
-rwxr-xr-x  1 funamoto    24576 Oct 17 03:22 exfile1
-rw-r--r--  1 funamoto      290 Oct 25 05:34 exfile1.c
-rw-r--r--  1 funamoto      989 Oct 30 00:22 exfile1.o
-rw-r--r--  1 funamoto       58 Oct 28 23:18 exfile1.txt
-rw-r--r--  1 funamoto      282 Oct 17 07:33 exfile2.c ←─1画面表示
                                                         後中断する
 [--More--[Press space to continue, 'q' to quit.]←─ space ：表示継続
                                                     q   ：表示終了
-rw-r--r--  1 funamoto       44 Oct 28 23:24 exfile2.txt
-rwxr-x---  1 funamoto    24576 Jan 12 05:14 exfilea
-rw-r--r--  1 funamoto       40 Dec 13 04:55 exscript
-rwxr-xr-x  1 funamoto       67 Oct 30 00:32 infile
drwxr-xr-x  2 funamoto      512 Oct 30 02:36 link
drwxr-xr-x  2 funamoto      512 Oct 25 05:18 mk
-rw-r--r--  1 funamoto       40 Nov 14 06:58 outfileaa
-rw-r--r--  1 funamoto       27 Nov 14 06:58 outfileab
-rw-r--r--  1 funamoto       67 Oct 20 04:57 students.dat
-rw-r--r--  1 funamoto      453 Oct 20 04:59 total.awk
-rw-r--r--  1 funamoto       40 Nov 14 06:57 xaa
```

```
-rw-r--r--  1 funamoto        27 Nov 14 06:57 xab
%

%more -5 exfile.c⏎ ←———— exfile.c の内容を 5 行単位で表示する
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
 [--More--(24%)] ←———— 5 行表示して中断する
        int c,p;          (24%) は通算で表示された率
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
 [--More--(56%)]
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
 [--More--(89%)]
                        }
                }
                p=c;
        }
        exit(0);
 [--More--(99%)]
}
%
```

more

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

mv

ファイルの移動および名前変更

move files

書式

❶ mv [*options*] *file1 file2*
❷ mv [*options*] *dir1 dir2*
❸ mv [*options*] *names dir*

❶ファイルを*file1*から*file2*に移動（名前変更）する。*file2*が存在している場合には、これを削除した後に移動する。
❷ターゲットのディレクトリ（*dir2*）が存在しない場合に、ディレクトリを*dir1*から*dir2*に移動（名前変更）する。
❸指定したファイルまたはディレクトリ（*names*）をターゲットのディレクトリ（*dir*）へ、そのままの名前で移動する。

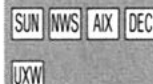
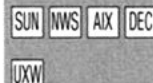
【引数】

file1　：移動元のファイル名
file2　：移動先のファイル名
dir1　：移動元のディレクトリ名
dir2　：移動先のディレクトリ名（存在しない）
names：1つ以上の移動元のファイル名またはディレクトリ名
dir　：移動先のディレクトリ名（存在する）

【オプション】

- ：これに続く引数をファイル名として解釈する。-ではじまるファイル名を指定できる。 SUN NWS AIX DEC UXW

-f：応答要求なしに、強制的に移動を実行する。

-i：既存のファイルがディレクトリである場合に、応答要求を出す。yではじまる行を入力したときだけ実行する。 SUN NWS AIX DEC UXW

【注】

・移動先が書き込み禁止モードとなっている場合には、プロンプトが出力され標準入力からの1行を読み取る。この行がyで始まる場合にmvコマンドが実行される。

【使用例】

```
%ls⏎
exfile.c   exfile1.c  exfile2.c
%mv exfile.c exfile3.c⏎ ←—— exfile.c を exfile3.c に名称変更する
%ls⏎
exfile1.c  exfile2.c  exfile3.c
%mkdir csub⏎ ←———————— サブディレクトリ csub を作成する
%mv *.* csub⏎ ←——————カレントディレクトリの全ファイルを csub に移動する
%ls⏎
csub ←—————————————— カレントディレクトリにはファイルがない
%ls csub⏎
exfile1.c  exfile2.c  exfile3.c ←—— csub にファイルが移動している
%mv csub cdir⏎ ←———————— ディレクトリ csub を cdir に名称変更する
%ls⏎
cdir
%
```

mv

nice

低優先順位でのコマンドの実行

書式

nice ［*option*］ *cmd* ［*args*］

●指定したコマンド（*cmd*）の優先順位を変更して実行する。

【引数】

cmd ［*args*］：優先順位を変更するコマンド（*cmd*）と引数（*args*）

【オプション】

+*n* ：*n*だけ優先順位を低くさせる（1 ＜ *n* ＜ 20）。
省略時：*n*＝ 4（スーパーユーザは減少も指定できる）

【使用例】

```
%nice find / -name core -print &⏎ ←── findコマンドを低い優先度で実行する
[1] 394
%ps⏎
  PID TT STAT  TIME COMMAND
  388 p3 S     0:01 -csh (csh)
  394 p3 R N   0:00 find / -name core -print
  395 p3 R     0:00 ps
%
```

nroff
テキストの清書処理

nontypesetting runoff

SVR BSD SUN HPU NWS AIX DEC SOL UXW

書式

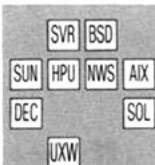
nroff [*options*] [*files*]

● NROFF形式で入力されたテキストを清書して出力する。

【オプション】(*)

- -o*list* ：指定したページ（*list*）のみ出力する。
- -n*n* ：最初のページ数（*n*）を指定する。
- -s*n* ：指定したページ（*n*）毎に停止する。
- -m*name* ：/usr/lib/tmac/tmac.*name* というマクロファイルを指定した入力ファイル（*files*）の前に挿入する。
- -r*an* ：レジスタ *a* に値（*n*）を設定する。（*a* は一文字）
- -i ：入力ファイルを読み込んだ後、標準入力から読み込みを行う。
- -q ：rdリクエストの同時入出力モードを呼び出す。

【使用例】

```
%cat extext⏎ ←———— NROFF形式で記述されたファイルの内容を表示する
.TL ←———— タイトル
Sample text for NROFF
.AU ←———— 著者名
Susumu Funamoto
.AI ←———— 著者の所属
Nippon Electronics College
Artificial Intelligent Laboratory
.AB ←———— アブストラクト開始
This is a sample text.
It will help for you to understand what is "NROFF"
.AE ←———— アブストラクト終了
.NH ←———— 見出し
Introducation
```

nroff

```
.PP ←―――――――― 新しい段落の開始
Nroff formats test in the named files for typewriter-like device.
Options may appear in any order so long as they appear before the files.
%nroff -ms extext > extext.nr⏎ ←―――― extextを清書したうえでファイル
%                                       extext.nrに出力する
```


(出力例)

extext.nr

```
                    Sample text for NROFF


                      Susumu Funamoto


                Nippon Electronics College
              Artificial Intelligent Laboratory



                         ABSTRACT

            This is a sample text.  It will help for  you
        to understand what is "NROFF"




     1.  Introducation

          Nroff formats test in the named files  for  typewriter-
     like  device.   Options  may  appear in any order so long as
     they appear before the files.
```

od

ファイルのダンプ

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

octal dump

書式

od [*option*] [*file*] [[+] *offset* [.] [b] [*label*]]

●指定したファイル（*file*）をダンプ（8進、10進、16進、アスキー）する。

【引数】

file　：ファイル名
　省略時：標準入力から読み込む。

+　：ファイル名(*file*)を省略した時に、オフセット値(*offset*)の前に付ける。

offset　：ダンプを開始する先頭から8進数のオフセット値

.　：オフセット値を10進数と見なす。

b　：オフセット値を512バイト単位のブロック数と見なす。

label　：ダンプ開始の擬似アドレス　SUN AIX DEC

【オプション】（*）

-b　：1バイト毎に8進数で出力する。

-c　：1バイト毎にアスキーコードで出力する。

-d　：2バイト毎に符号なしの10進数で出力する。

-o　：2バイト毎に8進数で出力する。

-s　：2バイト毎に符号付きの10進数で出力する。　AIX EWS OMR UXW

-x　：2バイト毎に16進数で出力する。

【使用例】

C言語のプログラムを通常の表示のほかに、アスキーおよび8進のダンプリストで表示する。

```
%cat exfile.c⏎
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
        int c,p;
        p='¥000';
        while((c=getchar())!=EOF){
                if (c!=DEL1 && c!=DEL2){
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                                putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
}
%od -c exfile.c⏎ ←―――――― アスキー形式による出力
0000000   #   i   n   c   l   u   d   e       <   s   t   d   i   o   .
0000020   h   >  ¥n   #   d   e   f   i   n   e       D   E   L   1
0000040   '   ¥   0   1   0   '  ¥n   #   d   e   f   i   n   e       D
0000060   E   L   2       '   ¥   1   3   7   '  ¥n   m   a   i   n   (
0000100   )  ¥n   {  ¥n  ¥t   i   n   t       c   ,   p   ;  ¥n  ¥t   p
0000120   =   '   ¥   0   0   0   '   ;  ¥n  ¥t   w   h   i   l   e   (
0000140   (   c   =   g   e   t   c   h   a   r   (   )   )   !   =   E
0000160   O   F   )   {  ¥n  ¥t  ¥t   i   f       (   c   !   =   D   E
0000200   L   1       &   &       c   !   =   D   E   L   2   )   {  ¥n
0000220  ¥t  ¥t  ¥t   p   u   t   c   h   a   r   (   c   )   ;  ¥n  ¥t
0000240  ¥t   }       e   l   s   e       {  ¥n  ¥t  ¥t  ¥t   i   f
0000260   (  ¥n  ¥t  ¥t  ¥t                   (   p   >   =   1   2   9
0000300       &   &       p   <   =   1   5   9   )  ¥n  ¥t  ¥t  ¥t
0000320               |   |       (   p   >   =   2   2   4       &   &
0000340       p   <   =   2   5   2   )   )   {  ¥n  ¥t  ¥t  ¥t  ¥t   p
0000360   u   t   c   h   a   r   (   c   )   ;  ¥n  ¥t  ¥t  ¥t   }  ¥n
0000400  ¥t  ¥t   }  ¥n  ¥t  ¥t   p   =   c   ;  ¥n  ¥t   }  ¥n  ¥t   e
0000420   x   i   t   (   0   )   ;  ¥n   }  ¥n
0000432
```

```
%od exfile.c      ←―――――――  8進ダンプリスト

0000000  021551 067143 066165 062145 020074 071564 062151 067456
0000020  064076 005043 062145 063151 067145 020104 042514 030440
0000040  023534 030061 030047 005043 062145 063151 067145 020104
0000060  042514 031040 023534 030463 033447 005155 060551 067050
0000100  024412 075412 004551 067164 020143 026160 035412 004560
0000120  036447 056060 030060 023473 005011 073550 064554 062450
0000140  024143 036547 062564 061550 060562 024051 024441 036505
0000160  047506 024573 005011 004551 063040 024143 020475 042105
0000200  046061 020046 023040 061441 036504 042514 031051 075412
0000220  004411 004560 072564 061550 060562 024143 024473 005011
0000240  004575 020145 066163 062440 075412 004411 004551 063040
0000260  024012 004411 004440 020040 020050 070076 036461 031071
0000300  020046 023040 070074 036461 032471 024412 004411 004440
0000320  020040 020174 076040 024160 037075 031062 032040 023046
0000340  020160 036075 031065 031051 024573 005011 004411 004560
0000360  072564 061550 060562 024143 024473 005011 004411 076412
0000400  004411 076412 004411 070075 061473 005011 076412 004545
0000420  074151 072050 030051 035412 076412
0000432
%
```

od

passwd

パスワードの設定または変更

password

書式

passwd [*options*] [*user*]

●ユーザ名（*user*）に対応したパスワードを設定または変更する。

【引数】

user：ユーザ名

省略時：現在のユーザ名が採用される。

【オプション】

-s　：ログイン時のシェルを変更する。　SUN NWS AIX SOL DEC UXW

-f　：finger コマンドで出力される情報を変更する。　SUN NWS AIX SOL DEC UXW

-F*file*：指定したファイル（*file*）をパスワードファイルとする。　SUN NWS DEC SOL PCU

【注】

・一般ユーザは自分のユーザ名に対応したパスワードだけ変更できる。その他については、スーパーユーザだけが許可される。

・パスワードには、自分の名前やユーザ名と同一の文字列など、連想しやすいものは避けるべき。

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

【使用例】

```
%passwd↵ ←──── ユーザ funamoto のパスワードを変更する
Changing password for funamoto on sun01.
Old password:______↵ ←──── すでに設定されているパスワード
                              未設定の場合は表示されない
New password:______↵ ←──── 新たに設定するパスワード
Retype new password:______↵ ←──── 新たに設定するパスワードの確認
%                                  （同一の文字列を入力する）
```

paste

複数ファイルの同一行の結合

SVR
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

書式

paste [*options*] *file1* [*file2*]

●指定したファイル（*file1 file2*）で同じ行を結合する。

【引数】

file1,*file2* ：結合するテキストファイル

【オプション】

-dlist ：入力ファイルの復帰改行文字を指定した文字(list)に置き換える。

-s ：隣接する行を結合する。

【使用例】

```
% cat exfile1 ⏎
Funamoto
Yamada
Abe
Ogawa
% cat exfile2 ⏎ ←――― exfile 2 と exfile 2 の内容を結合する
3711-1111
5467-1234
3369-9876
3211-4567
% paste exfile1 exfile2 ⏎
Funamoto 3711-1111
Yamada   5467-1234
Abe      3369-9876
Ogawa    3211-4567
% paste -s exfile1 ⏎
Funamoto Yamada Abe Ogawa
%
```

pg

テキストファイルの 1 ページ毎の出力

page

書式

pg [*options*] [*files*]

●指定したファイル（*files*）または標準入力の内容を 1 画面毎に出力する。

【引数】

files：出力するファイル名
　　省略時：標準入力から読み込む。

【オプション】

-*n*　：画面に出力する行数（*n*）を指定する。
-p *string*：プロンプトの文字列（*string*）を指定する。
　　省略時：':' をプロンプトとする。
-c　：各ページの表示前に画面をクリアする。
-e　：各ファイルの終わりで停止しない。
-f　：行の分割をしない。
-n　：改行を入力しなくてもコマンド入力されたと解釈する。
-s　：メッセージおよびプロンプトを反転して出力する。
+*n*　：開始行（*n*）を指定する。
+/*pattern*/　：指定したパターン（*pattern*）を含んでいる最初の行から出力する。

【使用例】

```
%ls -l /dev | pg⏎ ←—— /dev 以下のディレクトリ情報をページ単位で表示する
total 16
crw-rw-rw-   1 root   sys       44,  0 Dec  7 17:01 b4680
brw-rw-rw-   1 root   sys        2,  6 Jun  9  1989 bmem
crw-------   1 root   root      13,  0 Jun  9  1989 cal
drwxrwxr-x   2 root   sys           80 Jun  9  1989 clone
```

```
crw-------   3 root     sys          0,  0 Dec 13 13:47 console
crw-rw-rw-   1 root     root         1,  0 Jun  9  1989 cul0
crw-rw-rw-   1 root     root         1,  1 Jun  9  1989 cul1
crw-rw-rw-   1 root     root         1,  2 Jun  9  1989 cul2
drwxrwxr-x   2 root     sys            896 Jun  9  1989 dsk
cr--r--r--   1 root     sys          9,  0 Jun  9  1989 error
drwxrwxr-x   2 root     sys            224 Jun  9  1989 fd
c-w--w--w-   1 root     root        15,  0 Dec 13 11:51 fk
c-w--w--w-   1 root     root        15,  1 Jun  9  1989 fk1
c-w--w--w-   1 root     root        15,  2 Jun  9  1989 fk2
c-w--w--w-   1 root     root        15,  3 Jun  9  1989 fk3
c-w--w--w-   1 root     root        15,  4 Jun  9  1989 fk4
crw-rw-rw-   1 root     root        11,  0 Jun  9  1989 graph
crw-rw-rw-   1 root     sys          2,  5 Jun  9  1989 gv
crw-rw-rw-   1 root     sys          2,  6 Jun  9  1989 gv0
crw-rw-rw-   1 root     sys          2,  7 Jun  9  1989 gv1
crw-rw-rw-   1 root     sys          2,  8 Jun  9  1989 gv2
crw-rw-rw-   1 root     sys          2,  9 Jun  9  1989 gv3
: ⏎ ←———— 次のページを表示するときは改行を入力する
crw-rw-rw-   1 root     sys         34, 35 Jun  9  1989 jdict
crw-r--r--   1 root     sys          2,  1 Jun  9  1989 kmem
crw-rw-rw-   1 root     root        10, 16 Jun  9  1989 lp
crw-rw-rw-   1 root     root        10,  0 Jun  9  1989 lpt
drwxrwxr-x   2 root     sys            160 Jun  9  1989 md
crw-rw-rw-   1 root     root         1, 64 Jun  9  1989 mdm00
crw-rw-rw-   1 root     root         1, 65 Jun  9  1989 mdm01
crw-rw-rw-   1 root     root         1, 66 Jun  9  1989 mdm02
crw-r--r--   1 root     sys          2,  0 Jun  9  1989 mem
brw-r--r--   1 root     sys          0, 10 Nov  7 16:29 mshd00
brw-r--r--   1 root     sys          5, 10 Jun  9  1989 mshd20
crw-r--r--   1 root     sys         37,  0 Nov 16 12:34 nfsd
crw-rw-rw-   1 root     sys          2,  2 Dec 13 13:41 null
crw-rw-rw-   1 root     root        17,  0 Jun  9  1989 osm
crw-------   1 root     root         6,  0 Jun  9  1989 prf
crw-rw-rw-   1 root     sys         40,  0 Nov 17 14:02 ptyp0
crw-rw-rw-   1 root     sys         40,  1 Jun  9  1989 ptyp1
crw-rw-rw-   1 root     sys         40,  2 Jun  9  1989 ptyp2
crw-rw-rw-   1 root     sys         40,  3 Jun  9  1989 ptyp3
crw-rw-rw-   1 root     sys         40,  4 Jun  9  1989 ptyp4
crw-rw-rw-   1 root     sys         40,  5 Jun  9  1989 ptyp5
crw-rw-rw-   1 root     sys         40,  6 Jun  9  1989 ptyp6
:q⏎ ←———— 表示を中止するときはqを入力する
%
```

pg

popd

スタックからのディレクトリのポップアップ

CSH

popd

pop directory stack

書式

popd ［*option*］

●ディレクトリスタックの先頭を削除し、ワーキングディレクトリを２番目のエントリに変更する。

【オプション】

+*n*　：*n* 番目のエントリを削除する。
　　カレントディレクトリは変更されない。

【使用例】

```
%pwd↵         ← ワーキングディレクトリのパスを表示する
/usr/users/funamoto
%ls ~l↵       ← ワーキングディレクトリの情報を表示する
ex1    ex2    ← サブディレクトリ ex1 と ex2 が存在する
%pushd ex1↵   ← ワーキングディレクトリを ex1 に変更し、スタックの
~/ex1 ~          先頭にプッシュする。スタックの内容が表示される
%pwd↵
/usr/users/funamoto/ex1
%ls↵
c      dir     roff
%pushd c↵     ← ワーキングディレクトリを c に変更し、スタックの
~/ex1/c ~/ex1 ~  先頭に入れる
%pwd↵
/usr/users/funamoto/ex1/c
%popd↵        ← スタックの先頭にある~/ex1 をワーキングディレクト
~/ex1 ~          リにする
%pwd↵
/usr/users/funamoto/ex1
%popd↵        ← スタックの先頭にあるホームディレクトリをワーキン
~                グディレクトリにする
%pwd↵
/usr/users/funamoto
%
```

pr

プリンタ印刷用のフィルタ

SVR BSD SUN HPU NWS AIX DEC OMR SOL PCU UXW

書式

pr ［*options*］［*files*］

●指定したファイル(*files*)をプリンタ出力に適した形式でフォーマットして標準出力に書き出す。

【引数】

files　：フォーマットするファイル
　　省略時：標準入力から読み込む。

【オプション】

-f　：ページ区切りに用紙送りする。

-h *string*　：ヘッダとして指定した文字列(*string*)を出力する。

-l *n*　：1ページの行数(*n*)を指定する。
　　省略時：*n*=66

-m　：1カラム1ファイルで印刷する。

-s *c*　：空白の代わりに指定した文字(*c*)で区切る。

-t　：ヘッダとフッタの出力を省略する。

-w *n*　：1行の幅(*n*)を指定する。
　　省略時：*n*=72

+*n*　：*n*ページ目から印刷する。
　　省略時：*n*=1

-*n*　：*n*段にして出力する。
　　省略時：*n*=1

【使用例】

```
% pr exfile.src | lp ⏎
```

pr

printenv

環境変数の値の出力

print out the environment

書式

printev ［*var*］

●現在の環境変数の値を出力する。

【引数】

var ：環境変数名

省略時：すべての環境変数の値を出力する。

【注】

・環境変数については第3章Cシェルリファレンスを参照。

＜定義済み環境変数一覧＞

EXINIT	エディタの環境設定
HOME	ユーザのホームディレクトリ
LOGNAME	ユーザ名
PATH	コマンド実行時の環境対象となるディレクトリのリスト
PRINTER	プリンタ名のデフォルト値
PWD	ワーキングディレクトリ名
SHELL	ログイン時のシェル名
TERM	ターミナルのタイプ
TEMCAP	TERMに記述されている文字列
USER	ユーザ名

【使用例】

```
%printenv⏎ ←――――――― すべての環境変数の値を表示する
TERM=vt100 ←――――――― 端末名
HOME=/usr/users/funamoto ←――― ホームディレクトリ
SHELL=/usr/bin/csh ←――――― ログインシェルはCシェル
USER=funamoto ←――――――― ログイン名はfunamoto
PATH=/bin:/usr/ucb:/usr/users/funamoto/bin:/usr/bin ←― コマンド検索パス
LOGNAME=funamoto
PWD=/usr/users/funamoto ←――― ワーキングディレクトリ
EDITOR=/usr/bin/vi
MAIL=/usr/spool/mail/funamoto
LINK_TIMEOUT=3
EXINIT=set ai aw ic sw=4 redraw wm=4|map g G|map v ~~~~ ←― エディタ(vi)の環境設定
%
```

printenv

ps

現在のプロセスの状態出力

process status

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

書式

ps [*options*]

●現在のプロセスの状態を出力する。

SUN NWS AIX DEC

【オプション】(＊)

- a　：端末装置を使用中の全プロセスの状態を出力する。
- c　：コマンドの引数を出力しない。
- e　：コマンドの実行環境および引数を出力する。
- g　：全プロセスの状態を出力する。
- k　：カーネル仮想メモリの/dev/kmemとメモリの/dev/memの代わりに/vmcoreから情報を入手する。
- l　：詳細情報を出力する。
- s　：カーネル・スタック・サイズも出力する。
- t*n*　：指定した端末装置（*n*）のプロセスのみ出力する。
- u　：プロセスの所有者を出力する。
- v　：仮想メモリに関する情報も出力する。
- w*n*　：指定した幅（*n*）で出力する。
  *n*省略時：*n*＝132
- x　：端末装置を使用しない全プロセスの状態を出力する。
- #　：指定したプロセス番号（#）の状態を出力する。

EWS OMR PCU
HPU AIX UXW

【オプション】(＊)

- -e　：実行中の全プロセスに対する情報を出力する。
- -d　：プロセスグループのリーダを除く、全プロセスに対する情報を出力する。
- -a　：全プロセスに対する情報を出力する。
- -p*pids*：指定したID（*pids*）の情報を出力する。

【使用例】

```
%cc exfile.c &⏎
[1] 916
%ps⏎ ←――――――――― プロセスの状態を確認する
  PID TT STAT  TIME COMMAND
  765 p0 S     0:02 -csh (csh)
  916 p0 S     0:00 cc exfile.c
  918 p0 R     0:00 ccom - -fsoft -mc68020
  919 p0 R     0:00 ps
%

%ps gu⏎ ←――――――――― 所有者名も含めて全プロセスの状態を出力する
USER       PID %CPU %MEM   SZ  RSS TT STAT START   TIME COMMAND
funamoto   923  0.0  4.8  128  352 p0 R    15:52   0:00 ps gu
funamoto   765  0.0  4.8   56  352 p0 S    15:23   0:02 -csh (csh)
%ps a⏎ ←――――――――― すべてのプロセスの状態を表示する
  PID TT STAT  TIME COMMAND
  430 co T     0:00 cc -o test exfile.c exfile1.c exfile2.c exfile4.c
  438 co T     0:00 ccom - -fsoft -mc68020
  765 p0 S     0:02 -csh (csh)
  924 p0 R     0:00 ps a
  354 p1 I     0:00 -csh (csh)
%
```

PID　　　：プロセスID

TT　　　：端末

STAT　　：状態

（1文字目）

R……実行可能

T……停止状態

P……ページ待ち中

D……ディスク中で（または短い期間）待機中

S……20秒以内のスリープ

I……アイドル状態（20秒以上のスリープ）

（2文字目）

W……スワップアウト中

Z……削除されてはいないが停止中

TIME　　　：実行時間（CPU）

COMMAND：コマンド

USER　　　：プロセスの所有者

%CPU　　　：CPU使用率

%MEM　　　：実メモリ使用率

SZ　　　　：データセグメントとスタックセグメントの和（キロバイト）

RSS　　　：実メモリ（常駐セット）サイズ（キロバイト）

START　　：開始時刻

pushd

スタックへのディレクトリの格納

push directory stack

書式

pushd [*option*] [*dir*]

●ディレクトリ名をディレクトリスタックに格納する。カレントディレクトリも変更される。

【引数】

dir　：ディレクトリ名

【オプション】

+*n*　：*n* 番目のエントリをスタックの先頭に移動し、ワーキングディレクトリも移動する。
省略時：先頭と2番目のエントリを交換する。

【注】

・dirs コマンド (p.187) を参照。

CSH

PCU

pwd

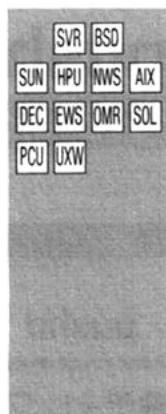
ワーキングディレクトリのパス名の出力

print current working directory

pwd

●現在のワーキングディレクトリのパス名を出力する。

【使用例】

```
%pwd⏎ ←―――――――― ワーキングディレクトリのパスを表示する
/usr/users/funamoto/c
%
```

rcp
リモートシステム間でのファイルのコピー

remote copy

BSD SUN NWS AIX DEC EWS OMR SOL UXW

書式

rcp ［*option*］ *file1 file2*
rcp ［*option*］ *files dir*

●ネットワークで結合されているマシン間でファイルのコピーを行う。

【引数】

file1、*file2*　：ファイル名
files　：1つ以上のファイル名
dir　：コピー先のディレクトリ名

【オプション】

-r：コピー元ファイルがディレクトリで指定されたとき、再帰的にサブディレクトリも含めてコピーする。

-p：コピー元ファイルのアクセス時刻、変更時刻、モードをそのままにしてコピーする。　SUN NWS AIX DEC EWS SOL UXW

【注】

・以上のリモートマシンのファイルおよびディレクトリの名前の指定は、

hostname:path

の形式で記述する。

ローカル側とリモート側でユーザ名が異なるときには、

username@hostname:path

の形式で記述する。

【使用例】

ローカルホスト sun07 のワーキングディレクトリにある exfile.c をネットワークで結合されたリモートホスト sun01 のディレクトリにコピーする。さらに、sun07 上で sun01 にコピーされた /tmp/exfile.c をワーキングディレクトリに exfile4.c という名前でコピーする。

```
%hostname⏎
sun07
%ls⏎
exfile.c        exfile1.c       exfile2.c       exfile3.c
%rcp exfile.c sun01:/tmp⏎ ←―――― リモートコピーの実行（sun07 → sun01）
%rcp sun01:/tmp/exfile.c ./exfile4.c⏎ ←―― リモートコピーの実行
                                            （sun01 → sun07）
%ls⏎
exfile.c        exfile1.c       exfile2.c       exfile3.c       exfile4.c
%
```

rlogin

リモートシステムへのログイン

remote login

BSD
SUN NWS AIX
DEC EWS OMR SOL
UXW

書式

rlogin [*options*] *hostname*
rlogin *hostname* [*options*]

SUN UXW
NWS AIX DEC OMR
EWS

●ネットワークで結合されているリモートマシン（*hostname*）にログインする。

【オプション】

-e*c*　：リモートマシンとの接続を切るエスケープコードを指定した文字（*c*）に変更する。
-l*user*：他のユーザ名（*user*）でログインする。
-L　：litout モードでログインする。
-8　：8 ビットコードを透過させる。
-7　：スタートおよびストップキャラクタが^S や^Q でない時以外にパリティビットを取り除く。

SUN NWS AIX EWS
OMR

【使用例】

ローカルホスト sun07 からリモートホスト sun01 にリモートログインする。

```
%hostname⏎ ←――――― 現在のホスト名を確認する
sun07
%rlogin sun01⏎ ←――― Sun01 にリモートログインする
Last login: Fri Sep  8 13:53:24 from vs201
Sun UNIX 4.2 Release 3.5EXPORT (PICKLE) #1: Mon Jun 26 15:31:50 PDT 1989
Sun Oct 29 16:00:48 JST 1989
%hostname⏎ ←――――― 現在のホスト名を確認する
sun01
%
```

rm

ファイルの削除

remove files

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

rm [*options*] *files*

●指定したファイル（*files*）を削除する。

【引数】

files：削除対象のファイル名またはディレクトリ名

【オプション】

- ：この後の引数をファイル名として処理する。

-f：ファイルに書き込み許可がなくても、問い合わせやメッセージの出力なしに削除する。

-r：指定したディレクトリ以下のすべてのファイル、サブディレクトリを再帰的に削除する。

-i：ファイル1つずつに対し、削除の可否を問い合わせる。

【使用例】

```
%ls⏎
exfile.c      exfile1.c      exfile2.c      exfile3.c      exfile4.c
%rm exfile4.c⏎ ←――――― ファイル exfile4.c を削除する
%ls⏎
exfile.c      exfile1.c      exfile2.c      exfile3.c
%rm -i *.*⏎ ←―― ワーキングディレクトリのファイルを1つ1つ削除の確認をする
rm: remove exfile.c? n ←―― n：削除しない
rm: remove exfile1.c? n
rm: remove exfile2.c? n
rm: remove exfile3.c? y←―― y：削除する
%ls⏎
exfile.c      exfile1.c      exfile2.c
%
```

rmdir

ディレクトリの削除

remove directories

SVR BSD SUN HPU NWS AIX EWS OMR SOL PCU

書式

rmdir *dirs*

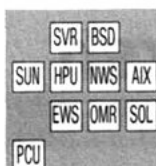
●指定したディレクトリ（*dirs*）を削除する。

【引数】

dirs：削除対象となるディレクトリ名
このディレクトリ以下にファイルやサブディレクトリが存在してはならない。

【使用例】

```
%ls -l⏎
total 6
drwxr-xr-x  2 funamoto      512 Oct 30  1989 c
drwxr-xr-x  2 funamoto      512 Oct 29 02:44 csh
drwxr-xr-x  2 funamoto      512 Oct 30  1989 exdir1
drwxr-xr-x  2 funamoto      512 Oct 30  1989 exdir2
drwxr-xr-x  2 funamoto      512 Oct 28 23:34 lisp
drwxr-xr-x  2 funamoto      512 Oct 29 04:53 work
%rmdir exdir1⏎ ←――――― ディレクトリ exdir1 を削除する
%ls -l⏎
total 5
drwxr-xr-x  2 funamoto      512 Oct 30  1989 c
drwxr-xr-x  2 funamoto      512 Oct 29 02:44 csh
drwxr-xr-x  2 funamoto      512 Oct 30  1989 exdir2
drwxr-xr-x  2 funamoto      512 Oct 28 23:34 lisp
drwxr-xr-x  2 funamoto      512 Oct 29 04:53 work
%
%rmdir work⏎ ←――――― ファイルをもつディレクトリは削除できない
rmdir: work: Directory not empty ←―― エラーメッセージが表示される
%
```

rsh

リモートシステムでのシェルの起動

remote shell

書式

① rsh ［*options*］ *hostname* ［*cmd*］

② rsh *hostname* ［*options*］ ［*cmd*］

●指定したコマンド（*cmd*）をローカルエリアネットワークで結合されたリモートマシン（*hostname*）で実行する。

【引数】

hostname：リモートマシンのホスト名

【オプション】

-l*user*：他のユーザ名（*user*）でリモートマシン上のコマンドの実行をする。
省略時：ローカルマシンのユーザ名と同じ

-n：標準入力を/dev/nullにリダイレクションする。

【注】

・rshは限定版シェル（rsh）を起動するコマンド。

【使用例】

```
sun01%rsh sun02 cc exfile2.c⏎   ← sun01からsun02に対して
%                                  コンパイルの依頼をする
```

ruptime

ローカルネットワーク上のマシンの稼働状況報告

BSD
SUN NWS AIX
DEC EWS OMR SOL
PCU UXW

remote uptime

書式

ruptime ［*options*］［*hostname*］

●ローカルエリアネットワーク上の各マシンの稼働状況を出力する。

【引数】

hostname：リモートマシンのホスト名 DEC

【オプション】

- -a ：1時間以上のアイドルユーザも対象とする。
- -l ：ロードアベレージでソートする。
- -r ：逆順にソートする。
- -t ：アップタイムでソートする。
- -u ：ユーザ数でソートする。
- -d ：ダウンしているマシンだけを出力する。 DEC
- -n ：指定したユーザ数（n）以上のマシンだけを出力する。 DEC

【使用例】

```
%ruptime⏎ ←―――― ネットワーク上のシステムのロードアベレージを表示する
sun01          up      7:14,     0 users,   load 0.01, 0.00, 0.00
sun02          up      7:14,     0 users,   load 0.01, 0.00, 0.00
sun03          up      7:14,     1 user,    load 0.34, 0.75, 0.57
sun04          down  2+21:24
sun05          down  2+21:25
sun06          down  2+21:26
sun07          up      7:16,     1 user,    load 1.28, 1.28, 0.80
sun08          down 48+21:50
sun09          down  6+19:53
sun10          up      7:15,     0 users,   load 0.00, 0.04, 0.00
sun11          down  4+20:07
sun12          up      6:12,     2 users,   load 0.00, 0.00, 0.00
sun13          up      1:02,     1 user,    load 0.14, 0.03, 0.00
sun14          up      6:12,     0 users,   load 0.00, 0.00, 0.00
sun15          down  4+19:49
sun16          up      6:11,     4 users,   load 0.20, 0.17, 0.01
sun17          up      2:05,     3 users,   load 0.00, 0.03, 0.00
sun18          down  4+20:38
sun19          down  4+20:16
sun20          up      0:26,     3 users,   load 0.16, 0.20, 0.00
vax01          up      7:12,     7 users,   load 1.85, 2.16, 2.16
vax02          up      7:13,     5 users,   load 0.84, 1.20, 1.27
vax03          up      7:13,     6 users,   load 0.42, 0.69, 0.74
vax04          up      7:12,     7 users,   load 2.64, 2.74, 2.64
vs201          up      7:16,     0 users,   load 0.23, 0.37, 0.25
%  ↑            ↑       ↑          ↑                ↑     ↑     ↑
 ホスト名      動作中   時間      ユーザ数          1分   5分   15分
              /停止中                               ロードアベレージ
```

ruptime

rwho

ローカルネットワーク上のユーザ情報報告

BSD
SUN NWS AIX
DEC EWS OMR SOL
PCU UXW

remote who

書式

rwho［*options*］［*users*］

●ローカルネットワークで結合されたマシンにログインしているユーザの情報を出力する。

【引数】

users ：指定したユーザ（*users*）に関する情報を出力する。 DEC

【オプション】

-a ：1時間以上のアイドルユーザも対象とする。

-h ：ホスト名でソートする。 DEC

rwho

【使用例】

```
%rwho⏎ ←――――――――――――― ネットワーク上のユーザ情報を表示する
user0012 sun20:console Dec 11 15:10 :24
user0012 sun20:ttyp0   Dec 11 15:10 :23
user0012 sun20:ttyp1   Dec 11 15:11
user0033 sun12:console Dec 11 15:13
user0042 sun16:console Dec 11 15:06 :18
user0042 sun16:ttyp0   Dec 11 15:17 :17
user0042 sun16:ttyp1   Dec 11 15:17 :17
user0042 sun16:ttyp2   Dec 11 15:17
user0042 vax03:ttyp0   Dec 11 13:31 :29
user0048 sun17:console Dec 11 07:24 :09
user0048 sun17:ttyp1   Dec 11 07:24 :09
user0048 sun17:ttyp3   Dec 11 07:25
user0058 sun13:ttyp0   Dec 11 15:25
user0072 vax01:ttyp1   Dec 11 15:04
user0079 vax04:ttyp0   Dec 11 15:19
user0082 sun12:ttyp0   Dec 11 15:19 :06
user1002 vax04:tty01   Dec 11 13:12 :01
user1005 vax04:tty00   Dec 11 13:15
user1006 vax04:tty02   Dec 11 13:15
user1010 vax01:tty10   Dec 11 13:19 :02
user1010 vax04:tty05   Dec 11 13:31
user1011 vax04:tty03   Dec 11 13:16 :02
user1012 vax03:tty02   Dec 11 13:11 :20
user1013 vax03:tty01   Dec 11 13:13 :01
user1014 vax03:tty00   Dec 11 13:10 :01
user1017 vax03:tty03   Dec 11 13:13
user1018 vax03:tty04   Dec 11 13:14
user1026 vax02:tty02   Dec 11 13:12
user1027 vax02:tty00   Dec 11 13:12
user1028 vax02:tty04   Dec 11 13:16 :04
user1032 vax02:tty01   Dec 11 13:13 :03
user1033 vax02:tty03   Dec 11 13:12
user1035 vax01:tty11   Dec 11 13:08 :02
user1057 vax01:tty13   Dec 11 13:16 :05
user1058 vax01:tty12   Dec 11 13:15
user1059 vax04:tty04   Dec 11 13:17
user1060 vax01:tty08   Dec 11 13:08
user1075 vax01:tty09   Dec 11 13:05 :01
%   ↑         ↑    ↑          ↑
ユーザ名  ホスト名：端末名    ログイン時刻
```

script

端末上でのセッションの記録

make typescript

BSD
SUN HPU NWS AIX
DEC EWS SOL
UXW

書式

script [*option*] [*file*]

●ユーザの端末に出力されるすべての情報を、指定したファイル（*file*）に格納または追加する。

【引数】

file ：出力を格納するファイル名

省略時：typescriptという名前のファイルに出力される。

【オプション】

-a ：追加モードで出力する。

-q ：メッセージ出力を抑制する。 EWS

-S *shell* ：シェルを絶対パス名で指定する。 EWS

script

script

【使用例】

```
%script⏎  ←—— これ以降の端末に対する出力を typescript ファイルに格納する
Script started, file is typescript
>> cd c⏎
>> pwd⏎
/usr/users/funamoto/c
>> ls⏎
a.out       exfile.c     exfile1.c     exfile2.txt   students.dat
cc          exfile.dat   exfile1.o     infile        total.awk
core        exfile.txt   exfile1.txt   link
exfile      exfile1      exfile2.c     mk
>> h 4⏎
    51  cd c
    52  pwd
    53  ls
    54  h 5
>> exit⏎
>> Script done, file is typescript
%cat typescript⏎
```

SVR

EWS

PCU UXW

sdb

シンボリックデバッガ

symbolic debugger

書式

sdb ［*option*］［*objfile* ［*corefile* ［*dirs*］］］

●指定された実行形式ファイル(*objfile*)をシンボリックデバックする。

【引数】

objfile ：実行形式ファイル名

コンパイル時に -g オプションを指定していれば、フルに sdb の機能を働かせることができる。

省略時：a.out が指定されたものとする。

corefile ：コアイメージファイル名

省略時：core が指定されたものとする。

dirs ：実行形式ファイルを作成する時に使われたソースファイルのあるディレクトリ（*dirs*）を指定する。

【オプション】

-w ：実行形式ファイルに対して書き込みを許可する。

-W ：ソースファイルが見つからない時や、ソースファイルが実行形式ファイルよりも新しいメッセージを出力しない。

sed
非会話型ストリームエディタ

SVR BSD SUN HPU NWS AIX DEC OMR SOL PCU UXW

書式

sed [*options*] [*files*]

●テキストファイル(*files*)を読み込み、スクリプトに従った編集を行い、結果を標準出力に書き出す。

【引数】

files　：編集対象のテキストファイル
省略時：標準入力から読み込む。

【オプション】

-e *script*　：スクリプト（*script*）を指定する。
-f *file*　：スクリプトをファイル（*file*）から読み込む。
-n　：この指定なしの時の出力を制限する。

【正規表現】

パターン	意　味
.	任意の1文字に一致
¥	特殊文字の機能を取り消し、通常文字として扱う。
*	この直前の文字またはグループの0回以上の繰り返しに一致
^	行の先頭と一致
$	行の末尾と一致
+	この直前の文字またはグループの1回以上の繰り返しに一致
?	この直前の文字またはグループの0または1回以上の繰り返しに一致
[*str*]	文字列(*str*)の内の1文字に一致
[*chr1*-*chr2*]	2つの文字(*chr1*,*chr2*)の範囲の1文字に一致
[^*str*]	文字列(*str*)の内の文字以外の文字に一致
(*expr*)	正規表現のグループ
expr1 | *expr2*	正規表現 *expr1* または正規表現 *expr2*

【使用例】

```
% cat -n exfile5 ⏎
     1  funamoto Susumu
     2
     3     ←――――――― 空白行
     4  funamoto Syotaro
     5
     6
% sed 's/funamoto/Funamoto/g' exfile5 ⏎ ←― funamoto という文字を
Funamoto Susumu                            Funamoto に置換する


Funamoto Syotaro



% sed '2,4 d' exfile5 ⏎ ←――― 2 行目から 4 行目までを削除する
funamoto Susumu


%
```

【注】

・-n オプションを指定したときに、必要な出力を得るには、sed のサブコマンド p を使う。

set

シェル変数値の出力または設定

CSH

set

書式

set ［*option*］

●シェル変数（*var*）の値の設定および値を出力する。

【オプション】

省略時	：現在設定されているすべてのシェル変数の値を出力する。
var	：シェル変数(*var*)の値を null をする。
var=*word*	：シェル変数(*var*)の値を *word* にする。
var[*n*]=*word*	：シェル変数(*var*)が配列の時、この *n* 番目の値を *word* にする。
var=*list*	：シェル変数(*var*)の値を順にリスト（*list*）で与えられた値にする。

【注】

・シェル変数の詳細は第3章Cシェルリファレンス(p.375)を参照。

＜定義済みシェル変数一覧＞

argv	シェルへの引数列
cdpath＝*paths*	ディレクトリ移動パス
cwd＝*path*	ワーキングディレクトリのパス
echo	エコーモード（トグル型）
histchrs *str1 str2*	ヒストリ置換文字
history＝*n*	ヒストリリストの行数
home＝*dir*	ホームディレクトリ
ignoreeof	EOFの無視（トグル型）
mail＝(*n mailfiles*)	メイルファイル
noclobber	ファイルへの出力制限（トグル型）
noglob	ファイル名の展開禁止（トグル型）
nonomatch	（トグル型）
notify	ジョブの完了通知（トグル型）
path＝*paths*	コマンド検索パス
prompt＝*string*	プロンプト
savehist＝*n*	ヒストリリスト・ファイルの行数
shell	シェルのファイル名
status	終了ステータス
time＝*n*	計時制御
verbose	エコーモード（トグル型）

【使用例】

```
%set⏎ ←―――――――― すべてのシェル変数の設定を表示する
argv    ()
cdpath  (/usr/users/funamoto/sys /usr/sys /usr/spool)
cwd     /usr/users/funamoto
history 50
home    /usr/users/funamoto
inc     /usr/include
mail    /usr/spool/mail/funamoto
notify
path    (. /bin /usr/ucb /usr/users/funamoto/bin /usr/bin)
prompt  %
savehist        50
shell   /bin/csh
status  0
term    vt100
user    funamoto
%set history=100⏎ ←―――― ヒストリリストを100に設定する
%set noclobber⏎ ←――――― 標準出力に制限を与える
%set⏎ ←―――――――――― 設定後の確認をする
argv    ()
cdpath  (/usr/users/funamoto/sys /usr/sys /usr/spool)
cwd     /usr/users/funamoto
history 100 ←――――――― 50から100に変更されている
home    /usr/users/funamoto
inc     /usr/include
mail    /usr/spool/mail/funamoto
noclobber ←―――――――― noclobberが新たに設定されている
notify
path    (. /bin /usr/ucb /usr/users/funamoto/bin /usr/bin)
prompt  %
savehist        50
shell   /bin/csh
status  0
term    vt100
user    funamoto
%
```

CSH

setenv

環境変数値の出力または設定

set environment variable

書式

setenv [*name* [*value*]]

●環境変数（*name*）の値の設定または値を出力する。

【引数】

name ：環境変数の名前

value ：新しい値の文字列

【使用例】

```
%setenv PRINTER lp1⏎ ← デフォルトのプリンタを指定するシェル変数にプリンタ名 lp1 を設定する
%printenv PRINTER⏎ ← シェル変数 PRINTER の設定内容を確認する
lp1
%
```

sh

標準シェルの起動

shell

書式

sh [*options*] [*arg*]

● Bシェルを起動する。

【オプション】(*)

- -i　：会話形式の入出力を行う。
  (端末に接続されているときと同様)
- -s　：コマンドを標準入力から読み込む。
  (引数が残っていないときも同様)
- -c *string*　：コマンドを文字列 (*string*) で読み込む。
- -r　：限定版シェルを起動する。

SVR BSD SUN HPU NWS AIX EWS OMR SOL PCU UXW

DEC EWS OMR SOL PCU UXW

【使用例】

```
%cat > exscript⏎ ←――― ファイル exscript にシェルスクリプトを記述する
who am i
date
^D
%sh exscript⏎ ←――― exscript を標準シェルで実行する
sun07!funamoto ttyp3   Mar  6 10:05     (cs102)
Tue Mar  6 10:06:54 JST 1990
%
```

size

オブジェクトファイルのサイズの出力

SVR BSD SUN HPU NWS AIX DEC EWS SOL PCU UXW

size of files

書式

size [*options*] [*files*]

●指定したオブジェクトファイル(*files*)の各セクション(TEXT, DATA, BSS)のサイズを出力する。

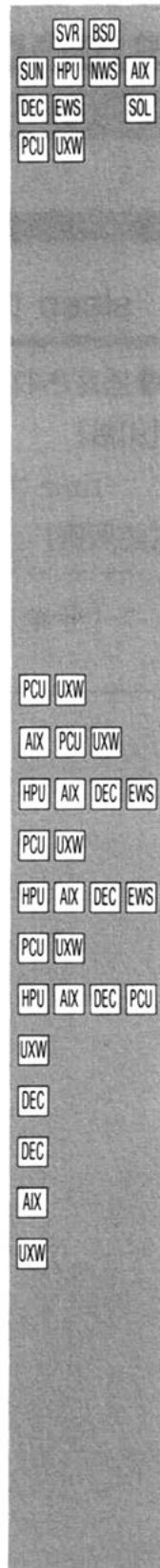
【引数】

files　：オブジェクトファイル名
　　　省略時：a.outが指定されたものとする。

【オプション】

-n　：NLOADセクションのサイズも出力する。 PCU UXW
-f　：セクション名も出力する。 AIX PCU UXW
-o　：8進数で出力する。 HPU AIX DEC EWS PCU UXW
-x　：16進数で出力する。 HPU AIX DEC EWS PCU UXW
-V　：sizeコマンドのバージョンを出力する。 HPU AIX DEC PCU UXW
-A　：SystemV形式で出力する。 DEC
-B　：4.3BSD形式で出力する。 DEC
-d　：10進で出力する。 AIX
-F　：各セグメントのサイズ、許可フラグ、合計サイズを出力する。 UXW

【注】

・各セクションの意味は次のとおり。
　TEXT：プログラムコード領域
　DATA：初期化されたextern, staticデータ領域
　BSS　：初期化されていないextern, staticデータ領域

sleep

コマンド実行の一時中断

書式

sleep time

●指定された時間（time）だけコマンドの実行開始を遅らせる。

【引数】

time：中断する時間（秒）

【使用例】

```
% (sleep 1200;cc exfile)& ⏎
```

← 20分後にコンパイルをバックグラウンドで実行する

sort

ファイルのソートまたはマージ

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

sort files

書式

sort ［*options*］［*files*］

●指定されたファイル(*files*)のすべての行をソートし、標準出力に書き出す。

【引数】

files　：ファイル名

【オプション】

-c　：指定した方法に従って入力ファイルがソートさているかをチェックし、ファイルがソートされていれば出力なし。

-m　：マージだけ行う。入力ファイルはソートされていなければならない。

-o*file*　：標準出力の代わりに使う出力ファイルの名称を file とする。入力のファイルの1つと同じファイル名でもよい。(-o*file* でも可)

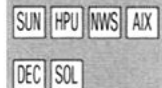
-T *dir*　：一時ファイルが作られるディレクトリ (dir) を指定する。　SUN HPU NWS AIX DEC SOL

-u　：複数の同一の内容の行があるとき、1行だけを残し、残りの行はすべて削除する。

-y*kmem*　：sort で使用するメモリ容量(*kmem*：キロバイト)を指定する。　SUN HPU AIX EWS OMR SOL PCU UXW

-z*recsz*　：バッファの容量(*recsz*)を指定する。-c、-m とともに使用する。　SUN HPU EWS OMR SOL PCU UXW

-b　：ソートキーのフィールドを決定する際に、先頭にあるブランク（スペースとタブ）を無視する。

-d　：'辞書'順にソートする。有効なキャラクタは文字、

数字およびブランクだけである。

-f　　：大文字を小文字と同様に扱う。

-i　　：非数値の比較において、ASCII コードの範囲(040 〜0176) 以外の文字を無視する。

-n　　：数字列に先行する、ブランク、マイナス記号および小数点が付いた 0、または任意の桁数の 10 進数から構成される数値数字列を、算術的な数値としてソートする。-b の働きも含む。

-r　　：逆順にソートする。

-t*c*　　：フィールドを区切るタブ文字を *c* にする。

-M　　：月順にソートする。最初の 3 文字で判定し、JAN<FBE<…<DEC の順。-b の働きも含む。

+*pos1* ［*pos2*］　：ソートキーのはじめの位置(*pos1*)と終わりの位置 (*pos2*) を指定する。終わりの位置を省略すれば、行の終わりとして処理する。

SUN HPU EWS OMR PCU UXW

【使用例】

```
%cat infile⏎ ←―― infileの内容を確認する
a m  80  95  60
b f  20  50 100
c m  40  40  40
d m 100  10  60
e f  20  60  90
%sort -r infile⏎ ←―― infileの内容を逆順に並べ替える
e f  20  60  90
d m 100  10  60
c m  40  40  40
b f  20  50 100
a m  80  95  60
%ls -l | sort -n +3⏎←ディレクトリリストをファイルの容量の昇順にソートする
total 226
-rw-r--r--  1 funamoto        27 Nov 13 16:57 xab
-rw-r--r--  1 funamoto        27 Nov 13 16:58 outfileab
-rw-r--r--  1 funamoto        30 Oct 28 10:27 exfile.dat
-rw-r--r--  1 funamoto        40 Nov 13 16:57 xaa
-rw-r--r--  1 funamoto        40 Nov 13 16:58 outfileaa
-rw-r--r--  1 funamoto        44 Oct 28 10:24 exfile2.txt
-rw-r--r--  1 funamoto        58 Oct 28 10:18 exfile1.txt
-rw-r--r--  1 funamoto        58 Oct 28 15:15 exfile.txt
-rw-r--r--  1 funamoto        67 Oct 19 15:57 students.dat
-rwxr-xr-x  1 funamoto        67 Oct 29 10:32 infile
-rw-r--r--  1 funamoto       282 Oct 16 18:33 exfile2.c
-rw-r--r--  1 funamoto       290 Oct 24 16:34 exfile1.c
-rw-r--r--  1 funamoto       296 Nov  1 13:04 exfile.c
-rw-r--r--  1 funamoto       453 Oct 19 15:59 total.awk
drwxr-xr-x  2 funamoto       512 Oct 24 16:18 mk
drwxr-xr-x  2 funamoto       512 Oct 29 12:36 link
drwxr-xr-x  3 funamoto       512 Oct 16 18:36 cc
-rw-r--r--  1 funamoto       989 Oct 29 10:22 exfile1.o
-rwxr-xr-x  1 funamoto     24576 Oct 16 14:22 exfile1
-rwxr-xr-x  1 funamoto     24576 Oct 28 15:45 exfile
-rwxr-xr-x  1 funamoto     24576 Oct 29 14:52 a.out
-rw-r--r--  1 funamoto   2188090 Oct 16 14:28 core
%                            ↑
                   ファイル容量の昇順になっている
```

sort

source

ファイル中のコマンド実行

define source for command input

CSH

SUN NWS

書式

source [*option*] *file*

●指定されたファイル (*file*) からコマンドを読み取り実行する。

【引数】

file ：実行するコマンドを記述したファイル名

【オプション】

-h ：コマンドを実行せずに、ヒストリリストに追加するだけ。

【使用例】

```
%cat.login⏎
stty dec new cr0
tset -I -Q
umask 022
setenv EDITOR '/usr/jsy/bin/vi'
setenv MAIL /usr/spool/mail/$USER
setenv  LINK_TIMEOUT    3
setenv SHELL /usr/jsy/bin/csh
setenv EXINIT 'set ai aw ic sw=4 redraw wm=4|map g G|map v ~~~~'
setenv OPSLIBRARY /usr/lib/ops5
set path = ($path /usr/lib/ops5)
set savehist=50
set mail=$MAIL
set prompt="%"
date
alias print 'print -T funamoto'
setenv TERM vt282
set term = "vt100"
stty erase ^H
%source .login⏎ ←――― 環境設定ファイル.loginの内容を実行する
Mon Dec 11 16:31:06 JST 1989
%
```

spline

スプライン曲線の補間座標の計算

spline curve

BSD SUN NWS AIX DEC EWS SOL PCU

spline

書式

spline ［*options*］

●標準入力から x,y 座標値を読み込み、スプライン曲線のための補間座標の組を出力する。

【オプション】

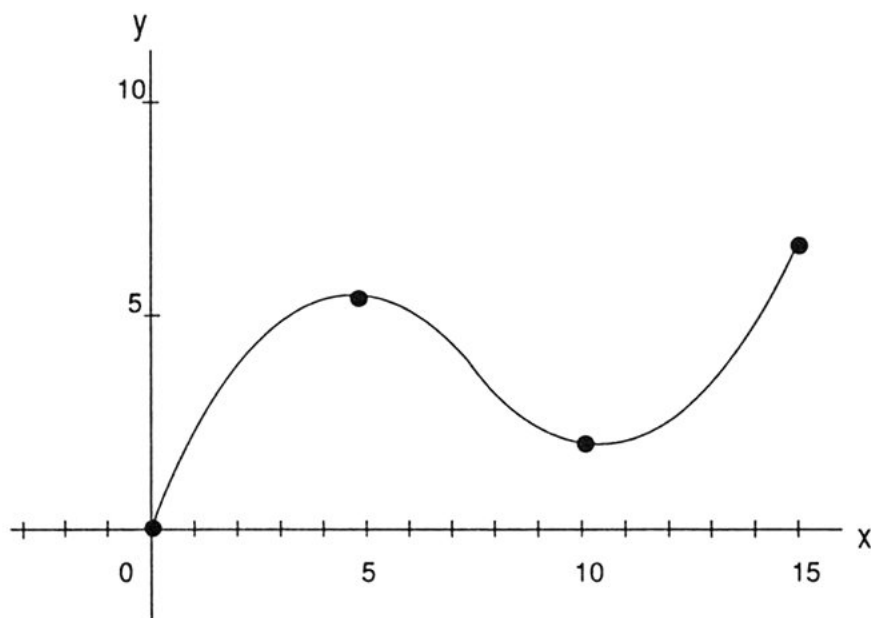
-a ：次の引数で座標間隔を決める。

-*k* ：$y''_0 = ky''_1$、$y''_n = ky''_{n-1}$
で境界値計算をする。定数 *k* を次の引数で指定する。

-*n* ：全区間を *n* 個に分割する。

-p ：周期的な出力を生成する。

-x ：次の引数で x の下限値を指定する。

【使用例】

```
%spline -n 15⏎ ←── 標準入力からスプライン曲線の構成点を4点入力し、
                    全区間を15分割した補間点の座標を得る
0 0⏎ ←┐
5 5⏎   └─ 標準入力から構成点の座標を入力する
10 3⏎
15 7⏎
^D ←── 入力を終了する
15.000000 7.000000  ┐
14.000000 5.803200  │
13.000000 4.705600  │
12.000000 3.806400  │
11.000000 3.204800  │
10.000000 3.000000  │
9.000000 3.240000   │
8.000000 3.768000   │
7.000000 4.376000   ├ 標準出力から補間点の座標が出力される
6.000000 4.856000   │
5.000000 5.000000   │
4.000000 4.652800   │
3.000000 3.870400   │
2.000000 2.761600   │
1.000000 1.435200   │
0.000000 0.000000   ┘
%
```

split

ファイルの分割

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR SOL
PCU UXW

split a file in pieces

書式

split [*option*] [*file* [*name*]]

●指定したファイル（*file*）を指定した数の行数に分割し、別のファイル（*name*）に出力する。

【引数】

file ：入力ファイル名
省略時：標準入力から読み込む。
(-を指定しても同様)

name ：出力ファイル名の接頭文字列
実際のファイル名は、*name* の後に aa、ab、……、zy、zz の文字を付加したもの(最大676個まで可能)。
省略時：出力ファイル名の接頭文字列を x とする。

【オプション】

-*n* ：行数（*n*）の指定。この行数単位で分割される。

【使用例】

```
%cat infile⏎ ←──────── 分割するファイルの内容を表示する
a m  80  95  60  ┐
b f  20  50 100  │
c m  40  40  40  ├ 5行のファイル
d m 100  10  60  │
e f  20  60  90  ┘
%split -3 infile outfile⏎ ←──── 先頭から3行目でファイルを分割する
%cat outfileaa⏎ ←──────── 前半3行のファイルの内容表示
a m  80  95  60
b f  20  50 100
c m  40  40  40
%cat outfileab⏎ ←──────── 後半2行のファイルの内容表示
d m 100  10  60
e f  20  60  90
%
```

stop

ジョブの停止

CSH

stop

書式

stop [*jobs*]

●指定したジョブを停止させる。

【引数】

jobs　：ジョブ番号

省略時：カレントジョブが指定されたものとする。

【使用例】

```
%cc exfile.c &⏎ ←―― C言語のプログラム exfile.c をバックグラウンドで
[1] 536                コンパイルする
%stop 536⏎ ←―――― バックグラウンドのジョブ 536 の実行を中断する


[1]  + Stopped (signal)    cc exfile.c ←― 中断したことを示すメッセージ
%fg⏎ ←――――――― フォアグラウンドで実行を再開する
cc exfile.c
%
```

stty
端末オプションの設定

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

set terminal port

書式

stty [*options*]

●現在の標準出力となっている端末装置のオプションを設定する。

【オプション】(*) SVR SUN HPU PCU UXW

-a ：すべてのオプションの設定を出力する。

-g ：別の stty コマンドに設定内容を引数として渡せる形式で現在の設定を出力する。

【オプション】(*) BSD NWS DEC

all ：機能キーのすべての確定状況を出力する。

everything：all の機能以外に付加情報を出力する。

以下の場合、指定の文字（*C*）がそれぞれの機能を果たす。 BSD NWS DEC UXW

erase *C* ：1 文字消去

kill *C* ：1 行消去

werase *C* ：1 単語消去

intr *C* ：強制割り込み

rprint *C* ：再表示

stop *C* ：出力の停止

start *C* ：出力の再開

eof *C* ：ファイルの終わり

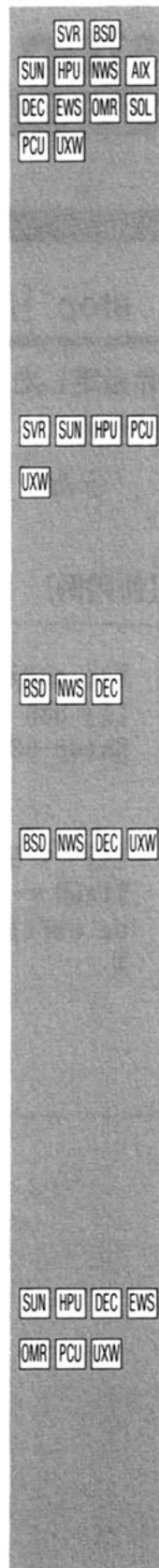
以下の場合、[-]は機能を抑止する。 SUN HPU DEC EWS OMR PCU UXW

[-]tabs ：タブをスペースに展開しない。

[-]echo ：全ての文字をエコーバックする。

[-]lcase ：大文字を小文字に見なす。

[-]nl ：行末を改行とする。

[-]even ：偶数パリティを許可する。

[-]odd　：奇数パリティを許可する。

[-]raw　：erase, kill の処理はしない。

[-]cbreak　：erase, kill 以外の入力をそのまま読み込む。

su

一時的なスーパーユーザまたは新ユーザへの切り替え

書式

su [*options*] [*name*]

●ユーザ id を一時的に他のユーザ（*name*）に変更する。

【引数】

name ：変更する他のユーザ名
省略時：スーパーユーザ(root)が指定されたものとする。

【オプション】

- \- ：指定したユーザのログイン時の環境が設定される。
- -f ：.cshrc を実行しない。 SUN NWS DEC
- -c *cmd* ：指定したユーザとしてログインした後にコマンド(*cmd*)を実行する。 SUN NWS PCU

【使用例】

```
% whoami⏎ ←———— 現在のログイン名を表示する
ueno
% su funamoto⏎ ←———— ユーザ名 funamoto でログインする
Password:______⏎
% whoami⏎ ←———— 現在のログイン名の確認
funamoto
%
```

tail

ファイルの末尾部分の出力

tail of file

書式

tail [*options*] [*file*]

●指定したファイル（*file*）の指定された位置以降を出力する。

【引数】

file ：ファイル名

省略時：標準入力から読み込む。

【オプション】

+*num* ：出力の開始位置は、ファイルの先頭からの指定による。

-*num* ：出力の開始位置は、ファイルの末尾からの指定による。

l ：行数による指定

b ：ブロック数による指定

c ：文字数による指定

r ：ファイルの末尾からの逆順に出力する。 SUN NWS DEC SOL UXW

f ：入力ファイルがパイプではない時に、入力ファイルの出力後に1秒間停止し、さらに入力ファイルから読み込んで出力する。

C ：全角および半角カナ文字も1文字として計算する。 EWS OMR

SVR BSD PCU NWS SUN OMR EWS AIX DEC HPU SOL

【使用例】

```
%cat -n exfile.c⏎ ←―――― ファイル exfile.c の内容を行番号を付けて表示する
     1  #include <stdio.h>
     2  #define DEL1 '¥010'
     3  #define DEL2 '¥137'
     4  main()
     5  {
     6          int c,p;
     7          p='¥000';
     8          while((c=getchar())!=EOF){
     9                  if (c!=DEL1 && c!=DEL2){
    10                          putchar(c);
    11                  } else {
    12                          if (
    13                              (p>=129 && p<=159)
    14                              || (p>=224 && p<=252)){
    15                                putchar(c);
    16                          }
    17                  }
    18                  p=c;
    19          }
    20          exit(0);
    21  }
%tail +10 exfile.c⏎ ←―――― 先頭から 10 行目以降を表示する
                        putchar(c);
                } else {
                        if (
                            (p>=129 && p<=159)
                            || (p>=224 && p<=252)){
                              putchar(c);
                        }
                }
                p=c;
        }
        exit(0);
}
%tail -5 exfile.c⏎ ←―――― 末尾から 5 行以降を表示する
                }
                p=c;
        }
        exit(0);
}
%
```

tail

talk
他のユーザとの会話

BSD SUN NWS AIX DEC EWS PCU UXW

書式

talk *user* [*tty*]

●指定したユーザ（*user*）と会話をする。

【引数】

user：会話相手のユーザ名

リモートマシン上のユーザの場合は、次の形式である。

host!user

host.user

host:user

user @ host

tty　：端末名

【使用例】

```
%talk user1⏎ ←―――――― ユーザ user1 と会話する依頼をする
```

ユーザ funamoto 側の画面

```
[Waiting for your party to respond] ←―― 相手方が応じるまで待状態になる




-----------------------------------------------------------------
```

ユーザ user1 の画面

```
%
.

Message from Talk_Daemon@sun07 at 14:46 ...
talk: connection requested by funamoto@sun07.
talk: respond with:  talk funamoto@sun07

%talk funamoto⏎
```

← ユーザ funamoto からの会話依頼のメッセージ

← ユーザ funamoto と会話する

ユーザ funamoto 側の画面

```
[Connection established]
Hello !
How are you ?
```

自分側
funamoto

```
------------------------------------------------------------------
I'm fine.
```

相手側
user1

tar

ファイルのテープ保存またはテープからのファイルの復旧

tape file archiver

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

tar ［*key*］［*files*］

●ファイル（*files*）を磁気テープ等にセーブまたはリストアする。

【引数】

file：ファイル名

key：以下の文字を指定する。

c ：新テープを作成する。

r ：指定したファイルをテープの最後にセーブする。

x ：指定したファイルをテープからリストアする。

t ：指定したファイルを見つけた毎に、その名前を出力する。
指定がなければ、すべてのファイル名を出力する。

u ：指定したファイルがテープ上に存在しないか、修正された時にそのファイルを追加してセーブする。

【オプション】（＊）

v ：機能を意味する文字と処理したファイル名を出力する。t オプションと併用すれば、テープエントリ情報も出力する。

f ：次に指定する引数をディレクトリ値の代わりに記録媒体とする。

w ：ファイル名と処理を出力して、確認のための入力待ちをする。y 入力で処理し、それ以外で処理しない。

b ：次に指定する数をデフォルト値の代わりにブロッキング・ファクタとする。数を省略したときには、20 ブロックと解釈する。

l ：不正なリンクがあるときに、メッセージを出力する。

m：更新時刻を復元しない。取り出された更新時刻とする。

o：ディレクトリのオーナーやモードについての情報をアーカイブに出力する。 SUN HPU NWS DEC EWS OMR SOL

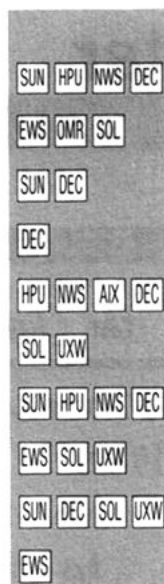
P：指定ファイルを元のモードに戻す。 SUN DEC

P：POSIX フォーマットを使用する。 DEC

p：指定ファイルを元のモードに戻す。 HPU NWS AIX DEC SOL UXW

h：シンボリックリンクをファイルやディレクトリと同様に扱う。 SUN HPU NWS DEC EWS SOL UXW

i：ディレクトリのチェックサム・エラーを無視する。 SUN DEC SOL UXW

M：複数ボリュームに対応する。 EWS

【使用例】

```
%tar cvf /dev/rst0 exfile⏎ ←――― ファイル exfile をテープにセーブする
a exfile 48 blocks
%tar tvf /dev/rst0⏎ ←――― テープの内容を確認する
rwxr-xr-x102/200  24576 Oct 29 04:45 1989 exfile
%tar xvf /dev/rst0⏎ ←――― テープの内容をリストアする
x exfile, 24576 bytes, 48 tape blocks
%
```

tee

パイプの接続

SVR BSD
SUN HPU NWS AIX
DEC OMR SOL
PCU UXW

書式

tee [*options*] [*files*]

●標準入力から読み込み、標準出力に書き出す。この際に指定したファイル（*files*）にもデータを格納することができる。

【引数】

files ：ファイル名

【オプション】

-i ：割り込みを無視する。

-a ：引数のファイルに対して追加書き込みする。

【使用例】

```
% cat exfile | tee outfile | lpr ⏎ ←exfile の内容を outfile に保存し、
                                     lpr コマンドに送る
```

tee

test

条件判定

書式

test *cond*

●指定した条件式(*cond*)を評価し、真なら0、偽なら1をシェル変数statusに設定する。

【条件式指定のオプション】

-d *file*	：ファイル(*file*)が存在し、ディレクトリであれば真
-f *file*	：ファイル(*file*)が存在し、ディレクトリでなければ真
-r *file*	：ファイル(*file*)が存在し、読み込み可能であれば真
-s *file*	：ファイル(*file*)が存在し、サイズが0以上であれば真
-w *file*	：ファイル(*file*)が存在し、書き込み可能であれば真
-n *str*	：文字列（*str*）の長さが0以上であれば真
-z *str*	：文字列（*str*）の長さが0であれば真
str1=*str2*	：文字列同士（*str1*,*str2*）が一致すれば真
str1!=*str2*	：文字列同士（*str1*,*str2*）が一致しなければ真
str	：文字列（*str*）がnullでなければ真
n1 *opr* *n2*	：2つの整数（*n1*,*n2*）を比較演算子（*opr*）で処理した結果が真であれば真

opr =-eq ：等しい
-ne ：等しくない
-gt ：大きい
-qe ：等しいか大きい

-lt　：小さい

-le　：等しいか小さい

【使用例】

```
% ls
testfile.txt
% test -f testfile.txt
% echo $status
0      ←―――――――――― 正常終了
% test testfile.dat
% echo $status
1      ←―――――――――― 異常終了
%
```

time

コマンドの実行時間の出力

SVR BSD SUN HPU NWS AIX DEC EWS OMR PCU UXW

書式

time ［*cmd*］

●指定したコマンド(*cmd*)の実行、および実行中に経過した時間、システム例で処理に要した時間、コマンド側で実行に要した時間を計測し、これを出力する。

【引数】

cmd：実行するコマンド行

【使用例】

```
%time cc exfile.c⏎ ←── ジョブ cc exfile.c の実行時間に関係する情報を表示する
1.4u 0.9s 0:03 61% 0+192k 3+17io 3pf+0w
%
```

- 0:03 … 経過時間
- 61% … コマンドの cpu 占有率

u…ユーザ使用の CPU 時間
s…システム使用の CPU 時間
k…プログラムの平均使用メモリ＋データの平均使用メモリ（キロバイト）
io…ディスクへのリード回数＋ライト回数
pf…ページフォルト回数
w…スワップ回数

tr

文字列の変換

translate characters

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

tr ［*options*］［*string1*］［*string2*］

●標準入力から読み込んだデータの中の指定した文字列（*string1*）を別の文字列（*string2*）に変換する。

【引数】

string1　：変換対象の文字列

string2　：変換文字列

【オプション】

-c　：8進文字コード001～377のうちで*string1*に含まれない文字を対象とする。

-d　：*string1*内の文字列に一致したものを削除する。

-s　：出力データの中で連続している文字列で*string2*の中の文字に一致している場合1文字にする。

【注】

・*string*の表記は次の形式を使うことができる。

［a-z］　：aからzまでのすべての文字を表す。

［a＊*n*］　：aが*n*個連続することを表す。

troff

写植機用の文書処理

SVR BSD SUN HPU NWS AIX SOL UXW

typesetting runoff

書式

troff ［*options*］［*files*］

●写植機に対して入力テキストを清書して出力する。

【オプション】

-o *list*　：指定したページ（*list*）のみ出力する。
-n*n*　：最初のページ数（*n*）を指定する。
-s*n*　：指定したページ（*n*）毎に停止する。
-m *name*：／usr／lib／tmac／tmac.name というマクロファイルを指定した入力ファイル(*files*)の前に挿入する。
-r*an*　：レジスタ *a* に値（*n*）を設定する。(*a* は一文字)
-i　：入力ファイルを読み込んだ後、標準入力から読み込みを行う。
-q　：rd リクエストの同時入出力モードを呼び出す。
-t　：写植機ではなく、標準出力に結果を出力する。
-f　：実行終了時に写植機を止め、用紙のフィードアウトをしない。
-w　：写植機が現在使用中の場合、使用可能になるまで待つ。
-b　：写植機が動作中か使用可能かを表示し、テキスト処理は行わない。
-a　：処理結果をプリント可能な ASCII コードに変換し、標準出力へ送る。
p*n*　：指定した値(*n*)のポイントサイズですべての文字を出力する。
-F *dir*　：指定したディレクトリ（*dir*）にフォント幅のテー

ブル/usr/lib/fontsが存在する。

【注】

・nroffコマンド（p.267）を参照。

tset

端末に関した初期設定

BSD SUN HPU NWS AIX DEC SOL UXW

terminal set

書式

tset [*options*]

●最初にUNIXにログインしたときに端末をセットアップする。

【オプション】（＊）

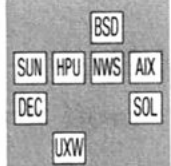
- -e*c* ：消去文字（*c*）をセットする。
  省略時：^H
- -k*c* ：行消去文字（*c*）をセットする。
  省略時：^X
- -i*c* ：割り込み文字（*c*）をセットする。
  省略時：^C
- \- ：最終的に決定された端末名を出力する。
- -r ：端末名を出力する。
- -n ：新しいttyドライバを指定する必要性の指定。
- -I ：端末初期設定ストリングの送信を抑制する。
- -Q ："Erase set to"および"Kill set to"のメッセージの印字を抑制する。

【使用例】

```
%tset⏎
Erase is Backspace ←―― 1文字削除はバックスペースに割り当てられている
%
```

tty

端末名の出力

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

terminal

書式

tty [*options*]

●ユーザの使用している端末のパス名を出力する。

【オプション】

-s：パス名を出力しない。終了ステータスだけの判定が必要なときに使用する。

終了ステータス＝0　　：標準入力が端末の時

0以外：その他の時

-l：ユーザ端末が使用可能な周期回線に接続されているときに回線名を出力する。 EWS OMR SOL PCU UXW

【使用例】

```
%tty ⏎ ←―――――――― 使用している端末名を表示する
/dev/ttyp0
%
```

unalias

コマンドの別名設定の解除

書式

unalias *strings*

●指定したパターン（*strings*）の別名をすべて無効にする。

【引数】

strings　：別名に一致させるパターン
　　　　　＊はすべてのパターンを意味する。

【注】

・alias コマンド（p.118）を参照。

【使用例】

```
% alias↵ ←――――――― 設定されているすべての別名を表示する
^[[A    clrs100
back    set back=$old; set old=$cwd; cd $back; unset back; dirs
cd      set old=$cwd; chdir !*
em      emacs
h       history
pd      pushd
pop     popd
print   print -T funamoto
pwd     echo $cwd
% unalias *↵ ←――――――― すべての別名を解除する
% alias↵
% ←――――――― 解除されている
```

uname

UNIXシステムの名称出力

name of current UNIX system

書式

uname ［*options*］

●現在使用しているUNIXのシステム名を出力する。

【オプション】（＊）

-s　：システム名を出力する。

-n　：ノード名を出力する。

-r　：OSのリリースを出力する。

-v　：OSのバージョンを出力する。

-m　：システムのハードウェア名を出力する。

-a　：すべての情報を出力する。

【使用例】

```
% uname ⏎  ←———— UNIXの名称を表示する
unix
%
```

uname

uniq
ファイル中の重複行の出力

書式

uniq [*options*] [*infile* [*outfile*]]

●ソートされたファイル中の重複している行を除いて出力する。

【引数】

infile ：入力のファイル名
　省力時：標準入力

outfile ：出力のファイル名
　省力時：標準出力

【オプション】

省略時：-ud

-u ：重複行以外だけを出力する。

-d ：重複行を1回だけ出力する。

-c ：各出力行の先頭に重複回数を付ける。

-*n* ：各行の先頭から *n* 個のフィールドをその前にある空白とともに無視する。

+*n* ：各行の先頭から *n* 文字を無視する。

【使用例】

```
% sort exfile6⏎
Funamoto
Susumu
funamoto
funamoto
susumu
syotaro
% uniq exfile6⏎
funamoto
Funamoto
susumu
syotaro
Susumu
funamoto
% sort exfile6 | uniq -d⏎
funamoto
% sort exfile6 | uniq -c⏎
   1 Funamoto
   1 Susumu
   2 funamoto
   1 susumu
   1 syotaro
%
```

CSH

unset

シェル変数値の設定の解除

書式

unset *vars*

●指定したパターンに一致したシェル変数を無効にする。

【引数】

vars：シェル変数名または一致させるパターン

＊はすべてのシェル変数を意味する。

【注】

・set コマンド (p.300) を参照。

【使用例】

```
% set⏎ ←――――――――――― すべてのシェル変数の設定内容を表示する
argv    ()
cdpath  (/usr/users/funamoto/sys /usr/sys /usr/spool)
cwd     /usr/users/funamoto
history 100
home    /usr/users/funamoto
ignoreeof
inc     /usr/include
mail    /usr/spool/mail/susumu
notify
path    (/bin . usr/users/funamoto/bin /usr/bin /usr/new /etc /usr/local/bin )
prompt  %
savehist        50
shell   /bin/csh
status  0
term    vt100
user    susumu
sun07% unset ignoreeof⏎ ←――――― ^D を無視するシェル変数 ignoreeof を解除する
sun07% set⏎ ←――――――――――― 解除されていることを確認
argv    ()
cdpath  (/usr/users/funamoto/sys /usr/sys /usr/spool)
cwd     /usr/users/funamoto
history 100
home    /usr/users/funamoto
inc     /usr/include
mail    /usr/spool/mail/susumu
notify
path    (/bin . usr/users/funamoto/bin /usr/bin /usr/new /etc /usr/local/bin )
prompt  %
savehist        50
shell   /bin/csh
status  0
term    vt100
user    susumu
sun07%
```

unsetenv

環境変数値の設定の解除

書式

unsetenv *vars*

●指定したパターンに一致する環境変数を無効にする。

【引数】

vars：環境変数名またはこれに一致させるパターン

＊はすべての環境変数を意味する。

【注】

・setenv コマンド（p.303）を参照。

【使用例】

uptime

時刻・稼働時間・ユーザ数・ロードアベレージの出力

BSD
SUN HPU NWS AIX
DEC
PCU UXW

show how long system has been up

書式

uptime

●システムが起動されてからの時刻、稼働時間・ユーザ数・ロードアベレージを出力する。

【使用例】

```
%uptime⏎
 11:17am  up  2:36,  4 users,  load average: 0.09, 0.02, 0.00
%
```

users

ユーザのログイン名の出力

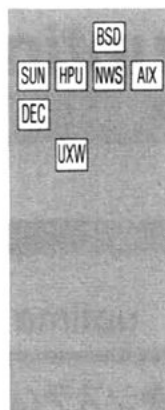
list of users

書式

users

●現在システム上に存在するユーザのログイン名を出力する。

【使用例】

vi

viエディタ(フルスクリーンエディタ)の起動

visual editer

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

書式

vi [*options*] *files*

●指定したファイル(*files*)を編集するためにフルスクリーンエディタviを起動する。exエディタを基本とした機能がある。

【引数】

files ：編集するファイル名

【オプション】

-l ：LISPプログラム用のセットアップをする。 SUN HPU NWS DEC OMR SOL UXW

-r ：エディタやシステムがクラッシュした時に復旧をする。

-t *tag* ：タグ（*tag*）を含むファイルを編集する。
*ctag*でタグが設定されているファイルが対象。

-w *n* ：ウィンドのサイズを*n*行にする。

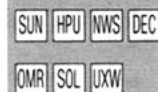
-*R* ：読み取り専用モードにする。

+*cmd* ：編集開始前にexコマンド（*cmd*）が実行される。

【注】

・viの使用法は第4章vi/emacsエディタ・コマンド・リファレンスを参照のこと。

【使用例】

```
%vi exfile.c⏎ ←―――― 既存のファイル exfile.c を編集する
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
```

```
        int c,p;
          p='¥000';
          while((c=getchar())!=EOF){
                  if (c!=DEL1 && c!=DEL2){
                          putchar(c);
                  } else {if ((p>=129 && p<=159) [14;29H|| (p>=224 && p<=252)){
                                          putchar(c);
                          }
                  } p=c;
          } exit(0);
}
~
~
"exfile.c" 21 lines, 282 characters
```

ファイル名 行数 文字数

```
%vi exfile0.c⏎
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"exfile0.c" [New file]
```

新規のファイル名を指定してエディタを起動する

view

viエディタの書き込み禁止モードでの起動

書式

view ［*options*］ *files*

●読み込み専用モードでviエディタを起動する。オプションやファイルの指定は、viコマンドに同一である。

【使用例】

```
%view exfile.c⏎ ←── ファイル exfile.c を書き込み禁止モードでエディタを起動する
#include <stdio.h>
#define DEL1 '¥010'
#define DEL2 '¥137'
main()
{
       int c,p;
         p='¥000';
         while((c=getchar())!=EOF){
                  if (c!=DEL1 && c!=DEL2){
                           putchar(c);
                  } else {if ((p>=129 && p<=159) [14;29H|| (p>=224 && p<=252)){
                                          putchar(c);
                           }
                  } p=c;
         } exit(0);
}
~
~
"exfile.c" [Read only] 21 lines, 282 characters ←── ファイル名：読み込み専用

"exfile.c" [File is read only] ←──── 書き込みを行ってセーブしようとすれば
                                      メッセージが表示される
```

view

w

現在ログイン中のユーザの実行コマンドの出力

BSD SUN NWS AIX DEC SOL PCU UXW

書式

w [*options*] [*user*]

●現在ログインしているユーザ名や実行しているコマンドなどのシステム状況を出力する。

【引数】

user：特定のユーザ名

省略時：全ユーザが対象

【オプション】（＊）

-h：見出しは出力しない。

-l：詳細情報を出力する（デフォルト）。

-s：省略情報を出力する。

-d：デバッグ情報を出力する。 DEC

-f：ログインしたホスト経路（from）を出力する。 DEC

-u：uptime コマンドと同じ情報を出力する。 DEC

【使用例】

現在ログインしているユーザの実行しているコマンドを表示する

現在の時刻　立ち上げてからの時間　ユーザ数　ロードアベレージ　1分　5分　15分

```
%w ⏎
 11:02am  up  2:21,  4 users,  load average: 1.00, 0.51, 0.17
User     tty       login@  idle   JCPU   PCPU  what
root     console  10:43am     1                -
funamoto ttyp0    10:56am            3      1  w
sumiyo   ttyp1    10:29am    14                -csh
tarou    ttyp2    10:43am     4     42     42  find /usr/users/student -name co
%
```

ログイン名　端末名　ログイン時刻　アイドル時間　全プロセスcpu時間　現在のプロセスcpu時間

W

wall

全ユーザへのメッセージの送信

SVR BSD
SUN HPU NWS AIX
DEC EWS OMR
PCU UXW

write to all users

書式

wall [*file*]

●全ユーザに対して標準入力から読み込んだメッセージを送る。

【引数】

file：送信したメッセージを格納したファイル名

SUN HPU

【オプション】

-g ：グループのユーザにメッセージを送る。

HPU UXW

【使用例】

ログインしている全ユーザにメッセージを送る。

```
%wall⏎ ←―――― 標準入力からメッセージを入力する
I am funamoto⏎ ←―――― 入力の終了
^D
Broadcast Message from root@sun07 (console) at 10:46 ...

I am funamoto
%



各ユーザの画面に表示されるメッセージ
%
Broadcast Message from root@sun07 (console) at 10:46 ...

I am funamoto
%
```

wc

ファイル中の文字数、単語数、行数の出力

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

word count

書式

wc [*options*] [*files*]

●指定したファイル（*files*）の行数、単語数、大文字を出力する。複数のファイルを指定した時には、全ファイルを通しての合計も出力する。

【引数】

files ：ファイル名

省略時：標準入力から読み込む。

【オプション】

省略時：-lwc を指定する場合と同じ。

-l：行数を出力する。

-w：単語数を出力する。

-c：文字数を出力する。

-b：バイト数を出力する。 SUN

【使用例】

```
%wc exfile.c exfile.txt⏎      ← ファイル exfile.c と exfile.txt の語数などの情報を表示する
      21      38     282 exfile.c
       4       8      58 exfile.txt
      25      46     340 total   ← 2つのファイルの合計
%
      ↑       ↑       ↑   ↑
      |       |       |   └── ファイル名
      |       |       └────── 文字数
      |       └────────────── 単語数
      └────────────────────── 行数
```

whatis

コマンドの簡単な説明の出力

BSD SUN NWS AIX DEC SOL UXW

書式

whatis *cmds*

●指定したコマンド（*cmds*）の簡単な説明（ヘッダライン）を出力する。man コマンドの -f オプションと同じ働きをする。

【使用例】

```
%whatis cb cc indent⏎ ←―――― cc、cb、indent コマンドの説明行を表示する
cb (1)                  - a simple C program beautifier
cc (1V)                 - C compiler
indent (1)              - indent and format a C program source file
%
```

whereis

プログラムのソース、オブジェクト、マニュアルの探索

書式

whereis ［*options*］ *files*

●指定したファイル(*files*)のソース、オブジェクト、マニュアル・セクションの所在を出力する。

【引数】

files：検索するファイル名

【オプション】

- -b ：バイナリファイルだけを検索する。
- -m ：マニュアル・セクションだけを検索する。
- -s ：ソースファイルだけを検索する。
- -u ：b、m、s以外のタイプのファイルを検索する。
- -B*dir* ：バイナリファイルを検索するディレクトリ（*dir*）を指定する。
- -M*dir* ：マニュアル・セクションを検索するディレクトリ（*dir*）を指定する。
- -S*dir* ：ソースファイルを検索するディレクトリ（*dir*）を指定する。
- -f ：B、M、Sのオプションを指定したときに*files*の指定がはじまることを示す区切り。

【使用例】

```
%whereis cc⏎ ←―――― ccコマンドの関連ファイルの所在を表示する
cc: /bin/cc /usr/bin/cc /usr/man/man1/cc.1v
%
```

which

コマンドの所在の表示

書式

which [*name***s**]

●指定した名前(*name***s**)で実行されるファイル名を表示する。alias や path での設定にも対応する。

【引数】

names：コマンド名

【使用例】

```
%which cc⏎ ←―― cc および mail コマンドの所在を表示する
/bin/cc
%which mail⏎
/usr/ucb/mail
%
```

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

who

現在ログイン中のユーザ名の出力

who is on the system

書式

who [*options*] [*file*] [am i]

●システムのユーザの情報を出力する。

【引数】

file ：情報を入手するファイル

省略時：／etc／utmp

am i：who コマンドを使用したユーザ自身の情報を出力する。

【オプション】

HPU EWS OMR PCU UXW

-u：現在ログインしているユーザを対象とする。

-T：端末回線の状態（出力する点）を除き、-u と同じ。

-l ：ログイン待ちの回線だけを出力。

-H：標準出力にヘッダを付ける。

-q：who の省略版。

-p：以前 init で生成され現在使用中の他のプロセスを対象とする。

-d：すでに終了し、init によって再生成されていないすべてのプロセスを対象とする。

-b：直前のリブートの日時を報告。

-r：init プロセスの現在の実行レベル。

-t：root によるシステム・クロックの最終変更。

-a：／etc／utmp または file に対してフルオプションにする。

-s：name、line、time フィールドだけ出力。

-A：utmp ファイル全体をカウントして報告する。

HPU AIX

【使用例】

%who
現在ログインしているユーザ名を表示する
root console Oct 29 10:24
funamoto ttyp0 Oct 29 10:21 (cs102)
susumu ttyp1 Oct 29 10:29 (vax01)
%
ログイン時刻
日付
端末名
ユーザ名

write

特定のユーザへのメッセージの送信

SVR BSD SUN HPU NWS AIX DEC EWS OMR SOL PCU UXW

write to another user

書式

write *user* [*tty*]

●標準入力から読み込んだメッセージを相手のユーザ（*user*）の端末に出力する。

【引数】

user：相手のユーザ名

tty ：相手が接続している回線名や端末名
複数の装置にログインしているユーザに対して使用する。

【注】

・他のユーザからの write によるメッセージの受信を拒否するには mesg コマンドを使用する。

・メッセージは 1 行単位で送信され、^D で受信する。

・ネットワークで接続されたリモートマシンのユーザに対して write コマンドを使うときには、ユーザ名にホストマシンの名称（*hostname*）を付加する。すなわち、

write *user*@*hostname*

とする。

【使用例】

特定のユーザ funamoto にメッセージを送る

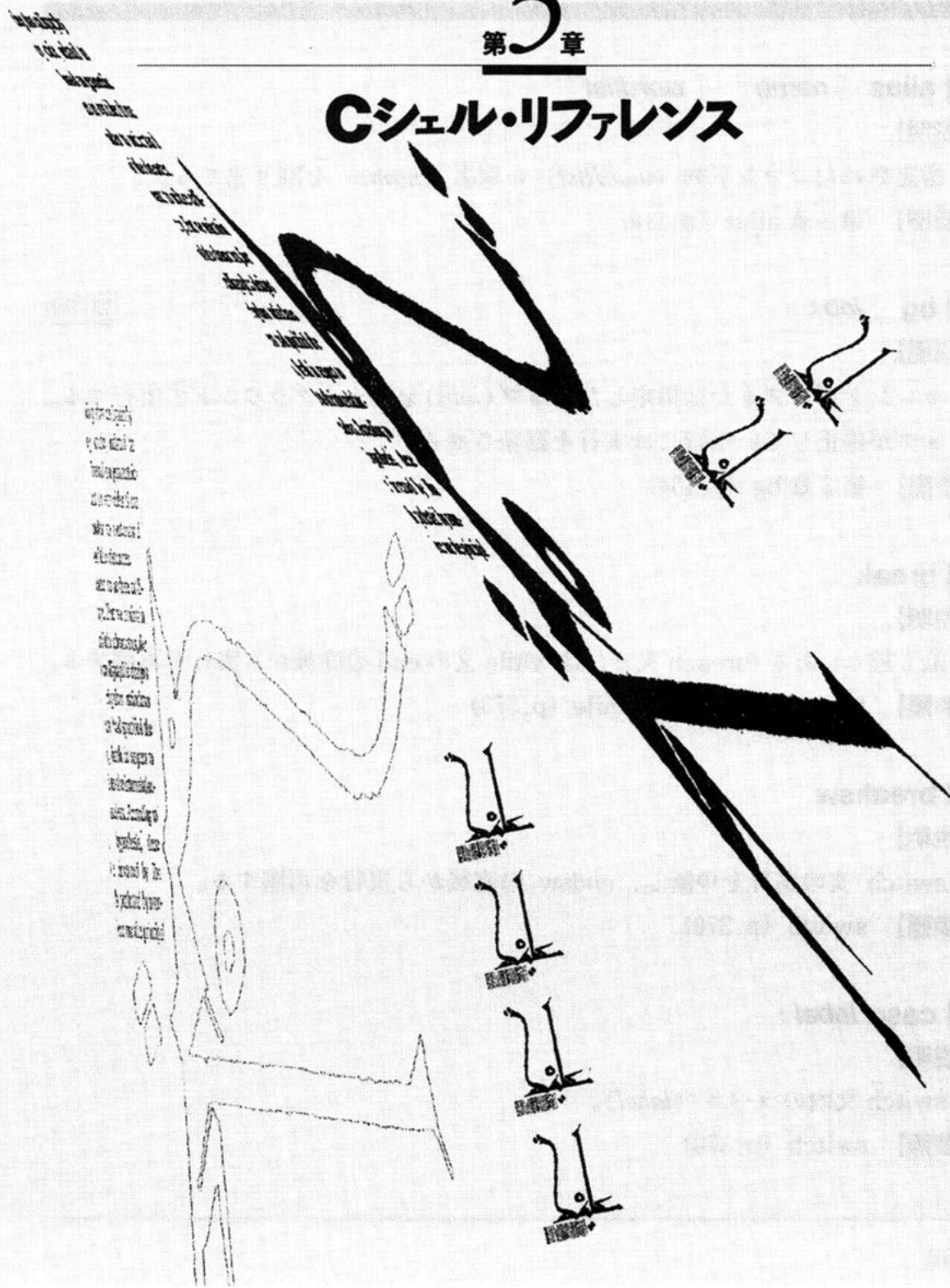
```
%write funamoto ⏎  ←── 標準入力に送信メッセージを入力する
I am funamoto ⏎  ←── 入力終了
^D
%
```

相手方に送信されるメッセージ

```
%
Message from funamoto@sun07 on ttyp0 at 10:19 ...
I am funamoto
EOF
%
```

write

第3章
Cシェル・リファレンス

3・1

Cシェルの組み込みコマンド

■ alias ［*name*］［*wordlist*］

【説明】

指定されたコマンド列（*wordlist*）に別名（*name*）を割り当てる。

【参照】 第2章alias（p.118）

■ bg ［*jobs*］ SUN NWS

【説明】

カレントジョブまたは指定したジョブ(*jobs*)をバックグラウンドで実行する。ジョブが停止している時には実行を継続させる。

【参照】 第2章bg（p.134）

■ break

【説明】

最も近くにあるforeach文またはwhile文のendの直後から実行を再開する。

【参照】 foreach（p.361）、while（p.373）

■ breaksw

【説明】

switch 文の実行を中断し、endsw の直後から実行を再開する。

【参照】 switch（p.370）

■ case *label* :

【説明】

switch文内のラベル（*label*）。

【参照】 switch（p.370）

■ cd ［*name*］
■ chdir ［*name*］

【説明】

ワーキングディレクトリを指定されたディレクトリ（*name*）に変更する。引数（*name*）が指定されない場合は、そのユーザのホームディレクトリに変更される。

【参照】 第2章cd（p.147）

■ continue

【説明】

最も近くにあるforeachまたはwhileループの実行を継続する。現在の行の残りのコマンドは実行される。

【参照】 foreach（p.361）、while（p.373）

■ default :

【説明】

switch文におけるデフォルトの場合のラベル。すべてのcaseラベルの後に記述する。

【参照】 switch（p.370）

■ dirs

【説明】

ディレクトリスタックを出力する。

【参照】 第2章dirs（p.187）

■ echo ［-n］ *wordlist*

【説明】

指定されたワード（*wordlist*）がスペースで区切られて標準出力へ書き出される。

【オプション】

　-n ：復帰改行をせずに終了する。

【参照】 第2章 echo (p.190)

■ eval *args*

SUN NWS

【説明】

引数が入力としてシェルへ読み取られ、結果として得られたコマンドがカレントシェルで実行される。

【例】

```
%set excsh1='ls -l'⏎ ←──── 変数 excsh1 に ls -l という値をセットする
%eval $excsh1⏎      ←──── 変数 excsh1 の値をコマンドとして実行する
total 28
-rwxr-xr-x  1 susumu      24576 Feb 16 04:23 a.out
-rw-r--r--  1 susumu        290 Oct 28 23:31 exfile1.c
-rw-r--r--  1 susumu        989 Feb 16 04:21 exfile1.o
-rw-r--r--  1 susumu        282 Oct 28 23:31 exfile2.c
-rw-r--r--  1 susumu        282 Oct 28 23:31 exfile3.c
%
```

■ exec *cmd*

【説明】

指定したコマンド（*cmd*）をカレントシェルに代えて実行する。

【例】

```
%exec ls -l⏎ ←──── ls コマンドを実行させた後にシェルを終了させる
total 28
-rwxr-xr-x  1 susumu      24576 Feb 16 04:23 a.out
-rw-r--r--  1 susumu        290 Oct 28 23:31 exfile1.c
-rw-r--r--  1 susumu        989 Feb 16 04:21 exfile1.o
-rw-r--r--  1 susumu        282 Oct 28 23:31 exfile2.c
-rw-r--r--  1 susumu        282 Oct 28 23:31 exfile3.c

login:←──── 自動的にカレントシェルが終了する
```

■ exit ［(*expr*)］

【説明】

現在のシェルを終了する。条件式 *expr* が省略されたときは、シェルは終了ステータスを status 変数の値とする。*expr* が指定されているときは、この値を終了ステータスとする。

【参照】 switch（p.370）

■ fg ［%*jobs*］ SUN NWS

【説明】

カレントジョブまたは指定されたジョブ(*jobs*)をフォアグラウンドで実行する。停止している時には実行を再開させる。

【参照】 第2章 fg（p.196）

■ foreach *name*（*wordlist*）

...

end

【説明】

wordlist の各要素を順番に変数 *name* にセットし、end との間にあるコマンドの並びを順番に実行する。

組込みコマンド continue を使用すると、ループを途中から継続することができ、組込みコマンド break を使用すると、ループを途中で終了させることができる。

【例】

3人のユーザ（user1 user2 user3）に hello world というメッセージを送る。

```
%cat excsh3⏎
#
foreach user (user1 user2 user3) ←── user1〜3の各ユーザにメールを送る
  echo hello world | mail $user
end
%excsh3
%
```

■ glob *wordlist*

【説明】

echoと同様の機能を果たす。ただし、'¥'エスケープは認識されず、ワード（*wordlist*）は出力ではヌル文字によって区切られる。

【例】 第2章echo（p.190）

■ goto *label*

【説明】

*label*で指定された行の直後から実行が継続される。

【参照】 switch（p.370）

■ history ［*option*］ ［*n*］

【説明】

ヒストリリストを表示する。*n*が与えられた場合には、最近の*n*個のヒストリだけが出力される。

【オプション】

-r：出力の順序を逆にして最新のものから順に出力する。

-h：各イベントの前に番号を付けずにヒストリリストを出力させる。

【参照】 第2章history（p.211）

■ if（*expr*）*cmd*

【説明】

指定された*expr*が真の場合には、引き数のある単一の*cmd*が実行される。

■ if (*expr1*) then

…

else if (*expr2*) then

…

else

…

endif

【説明】

指定された *expr1* が真であると、最初の else までのコマンドが実行される。*expr1* が偽の場合は、*expr2* が真なら 2 番目の else までのコマンドが実行される。else の部分は省略可。

【例】

現在の時刻から、それぞれに適当な挨拶のメッセージを出力する。

```
%cat excsh4⏎
#
set d=`date '+%H'`  ←── 変数 d に date コマンドの出力のうち時刻の値を
date                      セットする
if ($d<12) then ←──────── 変数 d の値が 12 より小さい時に
  echo good morning       good morning を出力
else if ($d<18) then ←─── 上記以外で変数 d の値が 18 より小さい時に
  echo good afternoon     good afternoon を出力
else ←─────────────────── さらに上記以外の場合
  echo good evening       good evening を出力
endif
%excsh4
Thu Mar 15 14:55:58 JST 1990
good afternoon
%
```

■ jobs [-l]

【説明】

現在実行中のジョブの状態を出力する。

【オプション】

-l：通常の情報に加えてプロセスIDも出力する。

【参照】 第2章jobs（p.219）

■ kill ［*-sig*］［*pids*］［*%jobs*］

【説明】

指定したジョブ（*pids, %jobs*）を強制終了させるか、またはジョブに対してシグナル（*sig*）を送る。

【参照】 第2章kill（p.223）

■ kill -l

【説明】

シグナルの名称を出力する。

【参照】 第2章kill（p.223）

■ login

【説明】

/bin/loginで置き換えて、ログインシェルを終了させる。ログオフする一つの方法。

【参照】 第2章login（p.234）

■ logout

【説明】

ログインシェルを終了させる。シェル変数ignoreeofがセットされていてCTRL+Dが使えないときに有効。

【参照】 第2章logout（p.236）

■ nice ［+*n* | -*n*］［*cmd*］

【説明】

シェルまたはコマンド（*cmd*）の優先度を*n*だけ増加させる。

-*n*のときには減少させる（スーパーユーザのみ）。

【参照】 第2章nice（p.266）

■ nohup ［ *cmd* ］

【説明】

引数が指定されていない時には、以降のシェルスクリプトでハングアップが無視される。コマンド（*cmd*）が指定されたときは、このコマンドについてハングアップを無視して実行される。

【参照】 第2章nice（p.266）

■ notify ［ *jobs* ］

【説明】

カレントジョブまたは指定されたジョブ（*job*）の状態が変化したときに、シェルが即時にユーザへ通知するようにする。

【例】

```
%cc exfile.c & ;notify⏎←──cc コマンドが終了次第メッセージが出力される
[1] 440
%

[1]     Done                    cc exfile.c ←──────終了メッセージの出力


[注] &はバックグラウンドジョブの指定
```

■ onintr ［ *label* ］

【説明】

割込みが発生したときに、指定したラベル（*label*）に制御を移す。ラベルを省略したときにはシェルスクリプトが終了する。'-'を指定すれば割込みを無視して処理を実行する。

【例】

```
%cat excsh6⏎
#
onintr intlabel◀――――割り込みが発生すればラベル intlabel に制御を移す

@ i=0
while($i<99)  ◀――――変数 i が 99 より小さい間は次の処理を繰り返す
  echo no interrupt ◀―― no interrupt を出力する
  @ i++
end
echo loop end
exit 0

intlabel:◀――――割り込みが発生した時
  echo interrupting◀―― intrerrupting を出力する

%excsh6
no interrupt
no interrupt
no interrupt
no interrupt
no interrupt
no interrupt
no interrupt
^Cinterrupting◀――――割り込み ctrl + C を発生させる
%
```


■ popd ［+*n*］

【説明】

ディレクトリスタックをポップし、新しいトップディレクトリに戻る。

【オプション】

+*n*：スタック内の *n* 番目（0から数える）のエントリを捨てる。

【参照】 第2章 popd（p.276）

■ pushd ［*name*］

指定したディレクトリ（*name*）をディレクトリスタックに入れて、これをワ

ーキングディレクトリとする。指定を省略した時は、先頭の2つのディレクトリを入れ換えて、新たに先頭になったものをワーキングディレクトリとする。

【参照】 第2章 pushd (p.283)

■ pushd [+*n*]

【説明】

ディレクトリスタックの *n* 番目のディレクトリを先頭にして、ここにワーキングディレクトリを移動する。

【参照】 第2章 pushd (p.283)

■ rehash

【説明】

シェルスクリプト内部のハッシュテーブルを再構成する。

【例】

```
%echo $path⏎ ←――――――――― コマンド検索パスを出力する
/usr/users/funamoto/bin  /usr/bin  /bin
%cp excsh /usr/users/funamoto/bin⏎ ←―― シェルスクリプト excsh を～/bin
%rehash⏎ ←―― ハッショテーブルに登録する       にコピーする
%
```

［注］ シェル変数 path については、3・2節（p.375）を参照

■ repeat *n cmd*

【説明】

指定されたコマンド（*cmd*）を指定回数（*n*）実行する。

【例】

```
%repeat 5 date⏎ ←――――――― date コマンドを 5 回連続実行する
Fri Mar  9 19:49:42 JST 1990
Fri Mar  9 19:49:42 JST 1990
Fri Mar  9 19:49:42 JST 1990
Fri Mar  9 19:49:42 JST 1990
Fri Mar  9 19:49:42 JST 1990
%
```

■ set ［*name* ［=*word*］］
■ set *name* ［*index*］=*word*

【説明】

シェル変数（*name*）に値（*word*）を設定する。引数を省略すれば、現在設定されているシェル変数の値を出力する。

【参照】 第 2 章 set（p.300）

■ setenv ［*name*］ ［*value*］

【説明】

環境変数（*name*）に値（*value*）を設定する。引数を省略すれば、現在設定されている環境変数の値を出力する。

【参照】 第 2 章 setenv（p.303）

■ shift ［*var*］

【説明】

シェル変数（*var*）のメンバが左へシフトされ、先頭の要素が破棄される。変数の指定がないときには、シェル変数 argv が対象となる。

【例】

入力された引数を１つずつ表示する。

```
%cat excsh8⏎
#
echo $argv
while ($#argv>0)     ←┘変数（argv）の個数が０より大きい間、以下の処理を
                         繰り返す
  shift         ←────── 変数（argv）のメンバを左に１つシフトする
  echo $argv    ←────── 変数（argv）の値を出力する
end

%excsh8 a b c⏎ ←──── ３つの変数 a,b,c, を指定して実行する
a b c
b c
c
%
```

［注］ シェル変数 argv については、３・２節（p.375）を参照

■ source ［-h］ *file*

【説明】

ファイル（*file*）に記述されたシェルスクリプトを現在のシェルで実行する。

【オプション】

-h：実行をせずに解釈だけを行う。ヒストリリストには書き出される。

【例】

個人用のシェル環境を設定するファイル~/.cshrc を更新したときに、これを有効にする。

```
%cat.login⏎
stty dec new cr0
tset -I -Q
umask 022
setenv EDITOR '/usr/jsy/bin/vi'
setenv MAIL /usr/spool/mail/$USER
setenv  LINK_TIMEOUT    3
setenv SHELL /usr/jsy/bin/csh
setenv EXINIT 'set ai aw ic sw=4 redraw wm=4|map g G|map v ~~~~'
setenv OPSLIBRARY /usr/lib/ops5
set path = ($path /usr/lib/ops5)
set savehist=50
set mail=$MAIL
set prompt="%"
date
alias print 'print -T funamoto'
setenv TERM vt282
set term = "vt100"
stty erase ^H
%source .login⏎ ←———————— 環境設定ファイル.loginの内容を実行する
Mon Dec 11 16:31:06 JST 1989
%
```

Cシェルの組み込みコマンド

■ stop ［*%jobs*］

【説明】

カレントジョブ、または指定されたバックグラウンドのジョブ（*%job*）を停止する。

【参照】 第2章stop（p.315）

■ suspend

【説明】

現在のシェルを停止する。

■ switch（*string*）

case ***str1:***

...

breaksw

...

default:

...

breaksw

endsw

【説明】

文字列 (*string*) の値によって、それぞれの処理を行う。該当するパターンがないときには、default：以下に記述した処理を行う。

【例】

```
%cat excsh10⏎
#
start: ←――――――――― goto 文に対応するラベル
echo "1:directory"           ┐
echo "2:users"               ├ メニューの出力
echo "9:exit"                ┘
echo -n "input number -> "
set innum=$<        ←―――― 標準入力からの値を変数 innum にセットする
switch($innum)      ←―――― 変数 innum の値によって処理を分ける
  case 1:                    ┐
    ls -l | more             ├ innum＝1 の時、ls コマンドを実行する
    breaksw                  ┘
  case 2:                    ┐
    who                      ├ innum＝2 の時、who コマンドを実行する
    breaksw                  ┘
  case 9:                    ┐
    exit                     ┘ innum＝9 の時、終了する
  default:                   ┐
    echo "input error"       ├ 上記以外の時、エラーメッセージを出力し、
    goto start               ┘ 再びメニューを出力する
endsw
%excsh10⏎
1:directory
2:users
9:exit
input number -> 2⏎
root     console Mar  9 19:54
funamoto ttyp0   Mar  9 19:49   (cs102)
%
```

Cシェルの組み込みコマンド

■ time ［*cmd*］

【説明】

引数が指定されない場合には、シェルとその子シェルが使用した時間の要約が出力される。

引数（*cmd*）が指定されれば、そのコマンドについて時間が要約が出力される。

【参照】 第2章 time（p.328）

■ umask ［*value*］

【説明】

ファイル作成マスクの出力または指定された値（*value*）の設定がされる。

■ unalias *pattern*

【説明】

指定されたパターン（*pattern*）に一致するすべての別名が取り消される。

【参照】 第2章 unalias（p.334）

■ unset *pattern*

【説明】

指定されたパターン（*pattern*）に一致する名前をもつシェル変数の設定を解除する。

【参照】 第2章 unset（p.338）

■ unsetenv *pattern*

【説明】

指定されたパターン（*pattern*）に名前が一致するすべての環境変数の設定を解除する。

【参照】 第2章 unsetenv（p.340）

■ wait

【説明】

すべてのバックグラウンドジョブの終了を待つ。

■ while (*expr*)

　　…

end

【説明】

expr が真の間、while と end の間に記述されたコマンド行を実行する。

break と continue は、それぞれループを途中で終了したり、途中から継続するのに使用する。

【例】

```
%cat excsh8⏎
#
echo $argv
while ($#argv>0)    ←  変数（argv）の個数が0より大きい間、以下の処理を繰り返す
  shift             ←  変数（argv）のメンバを左に1つシフトする
  echo $argv        ←  変数（argv）の値を出力する
end

%excsh8 a b c⏎     ←  3つの変数a,b,c,を指定して実行する
a b c
b c
c
%
```

［注］ シェル変数 argv については、3・2節（p.375）を参照

■ %*job*

【説明】

指定されたジョブ（%*job*）をフォアグラウンドへ切り替える。

■ *%job &*

【説明】

指定されたジョブ（*%job*）の実行をバックグラウンドで継続させる。

■@

■@ *name*＝*expr*

■@ *name*［*n*］＝*expr*

【説明】

変数（*name*）または変数の *n* 番目に値（*expr*）を設定する。

引数を指定しない場合には、すべてのシェル変数の値が出力される。

3・2

Cシェル変数

■ argv　シェルへの引数列

【説明】

シェルへの引数がセットされる。位置パラメータの内容がargvで置換されるため、$n は$argv[n]と同一の内容である。

【例】

```
%cat ex1⏎               ←――― シェルスクリプトの内容を表示する
#
echo $#argv             ←――― 引数の個数
echo $argv              ←――― 引数のすべての内容
echo $argv[1]           ←――― 1番目の引数の内容
echo $argv[2-4]         ←――― 2番目から4番目の引数の内容
echo $0                 ←――― ファイル名
echo $3                 ←――― 3番目の引数の内容
echo $*                 ←――― &argvと同じ
%csh ex1 a b c d⏎
4                       ┐
a b c d                 │
a                       │
b c d                   │ 各echoコマンドの出力結果
ex1                     │
c                       │
a b c d                 ┘
%
```

■ cdpath=*paths*　ディレクトリ移動パス

【説明】

ディレクトリを変更するコマンド（cd, chdir, pushd）の引数が、ワーキングディレクトリの下に存在しない時に、検索する候補のパスを与える。

【例】

```
%cd exdir⏎                                    ディレクトリの検索が不可能なため
exdir: No such file or directory ←── cd コマンドが失敗
%set cdpath=/usr/users/funamoto ←── ディレクトリ検索パスを指定
%cd exdir ←──────────────── cd コマンドが成功
~/exdir
%
```

■ cwd = *path*　ワーキングディレクトリのパス

【説明】

ワーキングディレクトリの絶対パス名

【例】

```
%echo $cwd⏎ ←──── ワーキングディレクトリのパスの出力
/usr/users/funamoto
%
```

■ echo　エコーモード（トグル型）

【説明】

コマンドが実行される直前にエコーする。

シェルを起動する際の-x オプションで設定される。第2章 csh(p.166)参照。

【例】

```
%set echo⏎ ←—— コマンドの実行直前に入力コマンドをエコーバックする
%date⏎
date          ←—— date コマンドのエコー
Thu Mar 15 17:04:19 JST 1990
%h 5⏎
history 5 ←—— alias の指定されているコマンドは展開されてエコーされる
    61  ls
    62  pwd
    63  set echo
    64  date
    65  h 5
%
```

［注］ alias コマンドについては、第２章（p.118）を参照

■ histchrs *string1 string2*　ヒストリ置換文字

【説明】

ヒストリ置換に使用する２つの文字

　１番目の文字（*string1*）…！に代わる文字

　２番目の文字（*string2*）…＾に代わる文字

【例】

```
%set histchars=";:" ←—— ヒストリ置換文字の！を；に、＾を：に変換する
%pwd
/usr/users/funamoto
%;; ←—— 直前のコマンドを実行する
pwd
/usr/users/funamoto
%
```

［注］ ヒストリ置換文字については、第２章 history コマンド(p.211)を参照

Cシェル変数

■ history = *n*　ヒストリリストの行数

【説明】

ヒストリリストに保存する行数

【例】

```
%echo $history⏎ ←――――― シェル変数 history の値を出力
100
%set history=50⏎ ←――――― シェル変数 history に 50 をセットする
%
```

■ home = *dir*　ホームディレクトリのパス

【説明】

ユーザのホームディレクトリ。~を展開する時にこの値を使う。

【例】

```
%echo $home⏎ ←――――― ホームディレクトリの値を出力する
/usr/users/funamoto
%
```

■ ignoreeof　EOFの無視（トグル型）

【説明】

入力装置からの EOF (end of file) の有効および無効を設定する。設定されていれば無効で[CTRL]+[D]によるシェルの終了を防ぐ。

【例】

```
%set ignoreeof⏎ ←――――― [CTRL]+[D]によってシェルが終了しないようにする
%^D                        [CTRL]+[D]の入力をしてもメッセージが出力
Use "logout" to logout.←―― されるだけ
%unset ignoreeof⏎    ←―― 解除する
%^D ←―――――――――――― [CTRL]+[D]でシェルは終了する

login:
```

■ mail=(*n mailfiles*)　メールファイル

【説明】

メールの着信をチェックするファイル名の設定する。

最初の語が数字であればチェックをする時間間隔（秒単位）と見なす。

　省略時：10分

設定されていれば着信時に

　you have a mail

を出力する。

【例】

```
%echo $mail⏎ ←――――――― メールファイルの所在を出力する
/usr/spool/mail/funamoto
%
```

■ noclobber　ファイルへの出力制限（トグル型）

【説明】

出力先に制限を与える。

　＞は新規ファイルのみに制限

　＞＞は既存ファイルのみに制限

【例】

```
%ls⏎
exfile
%set noclobber⏎ ←――――― ファイルへの出力を制限する
%cat > exfile⏎           } 既存ファイルへの標準出力はできない
exfile: File exists.     }
%cat >> newfile⏎                             } 新規ファイルへの追加モードの標準
newfile: No such file or directory           } 出力はできない
%unset noclobber⏎ ←――――― 解除する
%cat >> newfile⏎ ←――――― 新規ファイルへの追加モードの標準出力
                          ができる
a
b
c
^D
%ls⏎
exfile  newfile
%
```

［注］ 標準出力については、3・4節（p.389）を参照

■ noglob　ファイル名の展開禁止（トグル型）

【説明】

ファイル名の展開を禁止する。

【例】

```
%echo *file⏎
exfile newfile
%set noglob⏎ ←――――― ファイル名の展開を制限する
%echo *file⏎
*file
%
```

■nonomatch　ファイル名が一致しない場合にエラーメッセージを出さない指定(トグル型)

【説明】

ファイル名が既存のファイル名と一致しない場合にも、エラーにはしない。原形のファイル名のパターンを返却する。

【例】

```
%ls⏎
exfile.c        exfile1.c
%ls exfile*.lsp⏎
No match.
%set nonomatch⏎ ←── ファイル名が一致しなくてもエラーを出力しない指定
%ls exfile*.lsp⏎
exfile*.lsp not found
%
```

■ notify　ジョブの完了通知（トグル型）　SUN NWS

【説明】

ジョブの終了を非同期に通知する。

　省略時：プロンプトを表示する直前に通知する。

【例】

```
%set notify⏎ ←──── プロセスの状態が変化したことを即刻出力させる
%cc exfile.c &⏎
[1] 545
%

[1]    Done                cc exfile.c ←── cc コマンドの終了通知
```

■ path＝*paths*　コマンド検索パス

【説明】

コマンド実行時に探索対象となるディレクトリのリスト

初期設定：環境変数 PATH から *path* を設定。

pathが変更された時、PATHはCシェルによって自動的に更新される。
空語は、ワーキングディレクトリを指定したと見なす。

【例】

```
%echo $path⏎
/usr/ucb /bin . /usr/bin
%set path=(usr/users/funamoto/bin)⏎←── コマンド検索パスを変更する
%echo $path⏎
usr/users/funamoto/bin
%
```

■ prompt=*string*　プロンプト

【説明】

プロンプトとして表示する文字列（*string*）を指定する

　　省略時：%（一般ユーザ）

　　　　　　#（スーパーユーザ）

【例】

```
%set prompt="`hostname`% "⏎←──コマンドプロンプトをマシン名+%にする
sun01%
```

■ savehist=*n*　ヒストリリスト・ファイルの行数

【説明】

ログアウトの際に保存されるヒストリリストの数（*n*）を指定する。
ヒストリリストを保存するファイル。

　　~/.history

に保存される。

【例】

```
%echo $savehist⏎ ◀──────── 保存するヒストリリストの数を出力する
50
%ls ~/.history⏎
/usr/users/funamoto/.history ◀── ヒストリリスト保存ファイル
%
```

■ shell　シェルのファイル名

【説明】

シェルの存在するパス名

シェルスクリプトが実行されるときに起動するシェル。

【例】

```
%echo $shell⏎ ◀──────── 現在のシェル名を出力
/bin/csh      ◀──────── Cシェル
%
```

■ status　終了ステータス

【説明】

最後に実行されたコマンドのステータスを保持する。

　異常終了時：終了ステータスに0200を加えた値

組み込みコマンドの場合

　異常終了時：1

　正常終了時：0

【例】

```
%ls⏎
a.out            exfile.c         exfile1.c
%echo $status⏎
0 ◀──────────── lsコマンドが正常終了しているので0
%la⏎
la: Command not found.
%echo $status⏎
1 ◀──────────── lsというコマンドは存在しないので異常終了のため1
%
```

■ time＝*n*　計時制御

【説明】

コマンド実行時間の自動測定

時間（秒）を指定すれば、これ以上の時間を要したコマンドについて出力する。

表示される値の意味は、timeコマンドと同一である。

【例】

```
%set time=1⏎◀──────── 1秒以上の実行時間のかかったプロセスの情報を出力
%cc exfile.c⏎
1.3u 0.9s 0:04 50% 0+184k 3+20io 3pf+0w
%  合計時間が1秒以上
```

［注］ 出力情報の意味については、第2章timeコマンド（p.328）を参照

■ verbose　エコーモード（トグル型）

【説明】

ヒストリ置換実行後に各コマンドを出力する。

シェル起動時の-vオプションで設定される。第2章csh（p.166）参照。

【例】

```
%set verbose⏎
%ls ex*⏎
ls ex*
exfile.c        exfile1.c
%
```

3・3

コマンド実行時に使用する特殊文字

■; コマンドの連続実行

【説明】

複数のコマンドをセミコロン「;」で区切って、1行に記述することによって、左のコマンドから順番に連続実行される。

【例】

カレントディレクトリに新たにディレクトリ (dir1) を作成し、カレントディレクトリをそこに移し、新たなファイル名 (exfile1) を指定して vi エディタを起動する。

```
% mkdir dir1;cd dir1;vi exfile
```

■& コマンドの並列実行

【説明】

コマンド行の最後に&を記述することによって、コマンドはバックグラウンドジョブとして並列実行される。

【例】

C 言語のソースファイル (exfile1.c) をバックグラウンドでコンパイルする。

```
% cc -o exfile1 exfile1.c &
```

■| コマンドのパイプライン実行

【説明】

|の前のコマンドの標準出力を、あとのコマンドの標準入力に直結する。

【例】

ファイル (exfile1.c) の内容を1画面単位で出力する。

```
% cat exfile1.c | more
```

■ |& コマンドのパイプライン実行

【説明】

|&の前のコマンドの標準出力と標準エラー出力を、あとのコマンドの標準入力に直結する。

【例】

C言語のソースファイル(exfile1.c)をコンパイルし、この際のメッセージおよびエラーメッセージが多量である場合のために、1画面毎の出力をする。

```
% cc -o exfile1 exfile1.c |& more
```

■ ( ) コマンドをグループ化する

【説明】

;で区切られた複数のコマンドを( )で囲み、グループ化することによって、各コマンドに共通の指定をすることや、環境（カレントディレクトリ、ユーザ番号など）の変更を制限する。

【例】

psコマンドとwhoコマンドを連続実行し、この出力を1つのファイル(outfile)に書き出す。

```
%(ps;who) > outfile
%cat outfile
  PID TT STAT  TIME COMMAND
  504 p0 S     0:01 -csh (csh)
  520 p0 S     0:00 -csh (csh)
  521 p0 R     0:00 ps
susumu   ttyp0   Jun 13 17:51   (cs102)
%
```

特殊文字

カレントディレクトリを一時的にdir1に移動し、ここにあるファイル(exfile1.c)をコンパイルする。この後の状態としては、カレントディレクトリは元に戻っている。

```
%pwd
/usr/funamoto/dir
%(cd dir1;cc -o exfile exfile.c)
%pwd
/usr/funamoto/dir
%
```

■ &&, ||　コマンドの条件実行

【説明】

&&の前のコマンドが成功したときだけ、あとのコマンドが実行される。

||の前のコマンドが失敗したときだけ、あとのコマンドが実行される。

【例】

```
%find ~ -name core && echo found core file
%find ~ -name core || echo not found core file
```

3・4

標準入力(stdin)・標準出力(stdout)・標準エラー出力（stderr）などの切り替え

■ < 標準入力（stdin）リダイレクション

【説明】

標準入力を<のあとに記述したファイルに切り替える。

【例】

ファイル（infile）に記述したメッセージを、指定したユーザ（user1）にメールとして送信する。

```
% mail user1 < infile
```

■ > 標準出力（stdout）リダイレクション

【説明】

標準出力を>のあとに記述したファイルに切り替える。

【例】

カレントディレクトリの内容をファイル（exfile1）に書き出す。

```
% ls -l > outfile
```

■ >& 標準出力&標準エラー出力リダイレクション

【説明】

標準出力と標準エラー出力を>のあとに記述したファイルに切り替える。

【例】

C言語のソースファイル(exfile1.c)をコンパイルし、この際のすべての出力を1つのファイル（outfile）に書き出す。

```
% cc -o exfile1 exfile1.c >& outfile
```

■ >!　標準出力リダイレクション（シェル変数 noclobber の指定を無視する）

【説明】

既存ファイルに対しての出力に制限を加えている場合にも、強制的に標準出力を>! のあとに記述したファイルに切り替える。

【例】

カレントディレクトリの内容をすでに存在するファイル（outfile）に書き出す。

```
% ls -1 >! outfile
```

■ >&!　標準出力2&標準エラー出力リダイレクション（シェル変数 noclobber の指定を無視する）

【説明】

既存ファイルに対しての出力に制限が加えられている場合にも、強制的に標準出力と標準エラー出力を>&! のあとに記述したファイルに切り替える。

【例】

C言語のソースファイル(exfile1.c)をコンパイルし、この際のすべての出力をすでに存在する1つのファイル（outfile）に書き出す。

```
% cc -0 exfile1 exfile1.c >&! outfile
```

■ >>　標準出力リダイレクション（追加モード）

【説明】

標準出力を>>のあとに記述したファイルに切り替える。すでにファイルが存在する場合には、ファイルの最後に追加する。

■ >>&　標準出力&標準エラー出力リダイレクション（追加モード）

【説明】

標準出力と標準エラー出力を>>&のあとに記述したファイルに切り替える。すでにファイルが存在する場合には、ファイルの最後に追加する。

■ >>!　標準出力リダイレクション（追加モード、シェル変数 noclobber の指定を無視する）

【説明】

既存ファイルに対しての出力に制限が加えられている場合にも、強制的に標準出力を>>!のあとに記述したファイルに切り替える。すでにファイルが存在する場合には、ファイルの最後に追加する。

■ >>&!　標準出力&標準エラー出力リダイレクション（追加モード、シェル変数 noclobber を無視する）

【説明】

既存ファイルに対しての出力に制限が加えられている場合にも、強制的に標準出力と標準エラー出力を>>&!のあとに記述したファイルに切り替える。すでにファイルが存在する場合には、ファイルの最後に追加する。

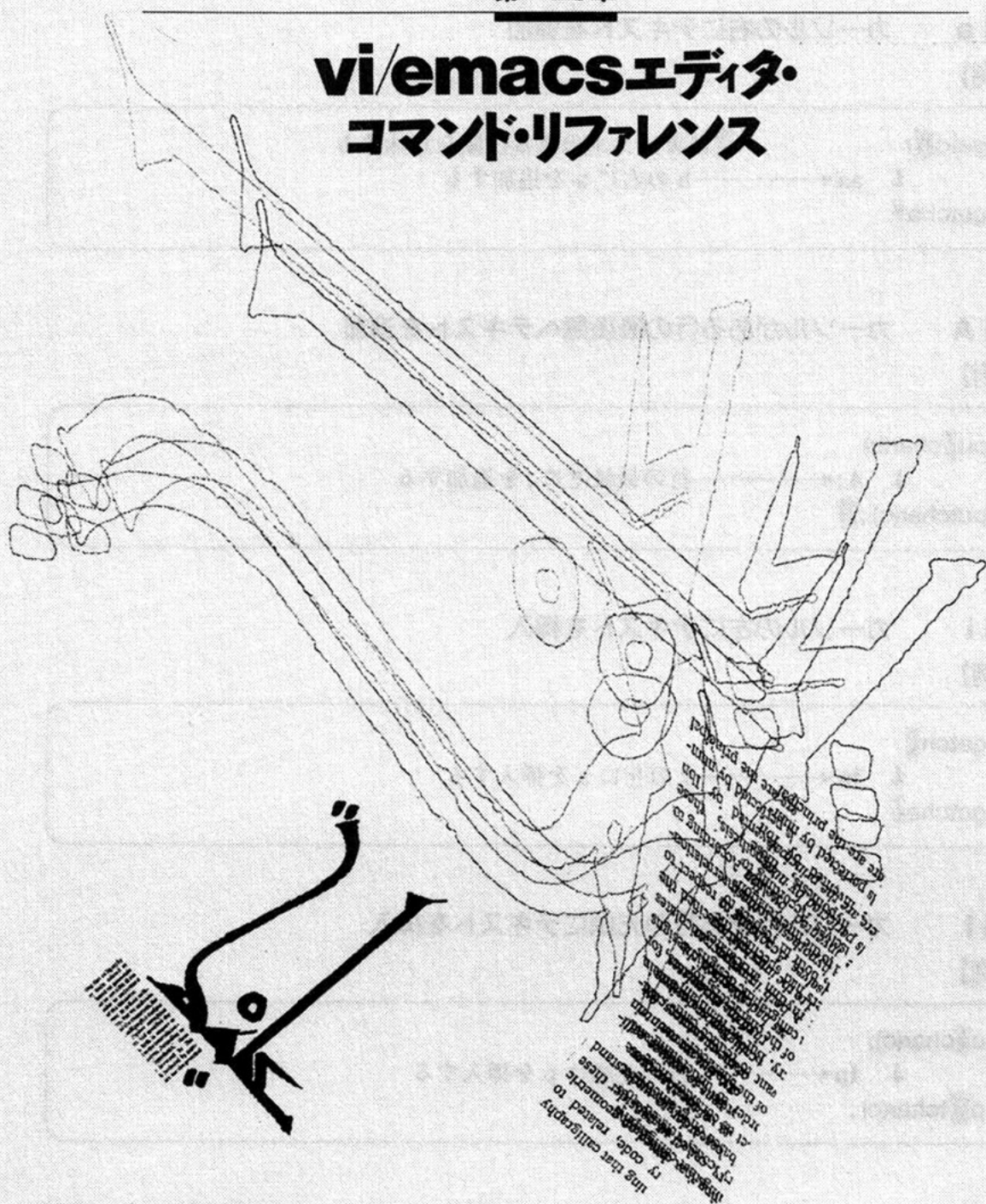
■ <<*arg*　標準入力リダイレクション

【説明】

行の先頭に *arg* で指定した文字列が現れるまで、入力行を読む。

第4章

vi/emacsエディタ・コマンド・リファレンス

4・1

追加・挿入（テキスト入力モードに入る）

■ a　　カーソルの右にテキストを追加

【例】

```
getchr              （h がカーソルのある位置）  ■はカーソルのある位置を意味する
      ↓ aa ←──── h の右に a を追加する
getchar
```

■ A　　カーソルがある行の最後尾へテキストを追加

【例】

```
putchar(c)
      ↓ A; ←──── 行の最後尾に ; を追加する
putchar(c);
```

■ i　　カーソルの左にテキストを挿入

【例】

```
getchr
      ↓ ia ←──── r の左に a を挿入する
getchar
```

■ I　　カーソルがある行の先頭にテキストを挿入

【例】

```
utchar(c);
      ↓ Ip ←──── 行の先頭に p を挿入する
putchar(c);
```

■ o　カーソルがある行の下の行にテキストを追加

【例】

```
putchar(c);
j=i;
        ↓  oi++; ←――――;行の下にi++;を追加する
putchar(c);
i++;
j=i;
```

■ O　カーソルがある行の上の行にテキストを挿入

【例】

```
putchar(c);
j=i;
        ↓  Oi++; ←――――行の上にi++;を追加する
putchar(c);
i++;
j=i;
```

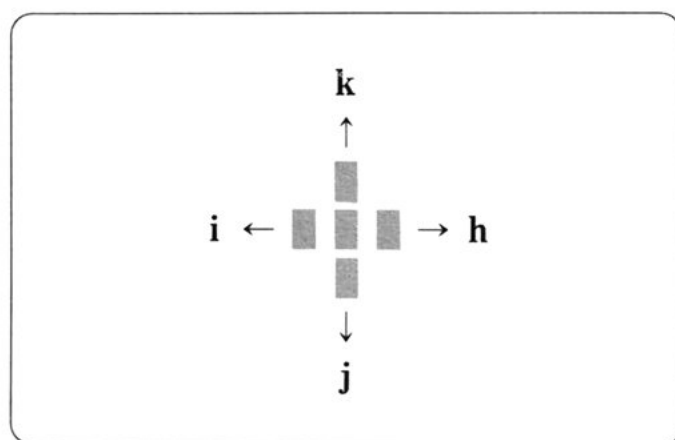
Emacs

Emacsにはモードの区別がないため、viの追加・挿入に対応するコマンドは必要ない。

追加・挿入

4・2

カーソルの移動・検索

■ ←または h　1文字左へ移動
■ ↓または j　1行下へ移動
■ ↑または k　1行上へ移動
■ →または l　1文字右へ移動

Emacs

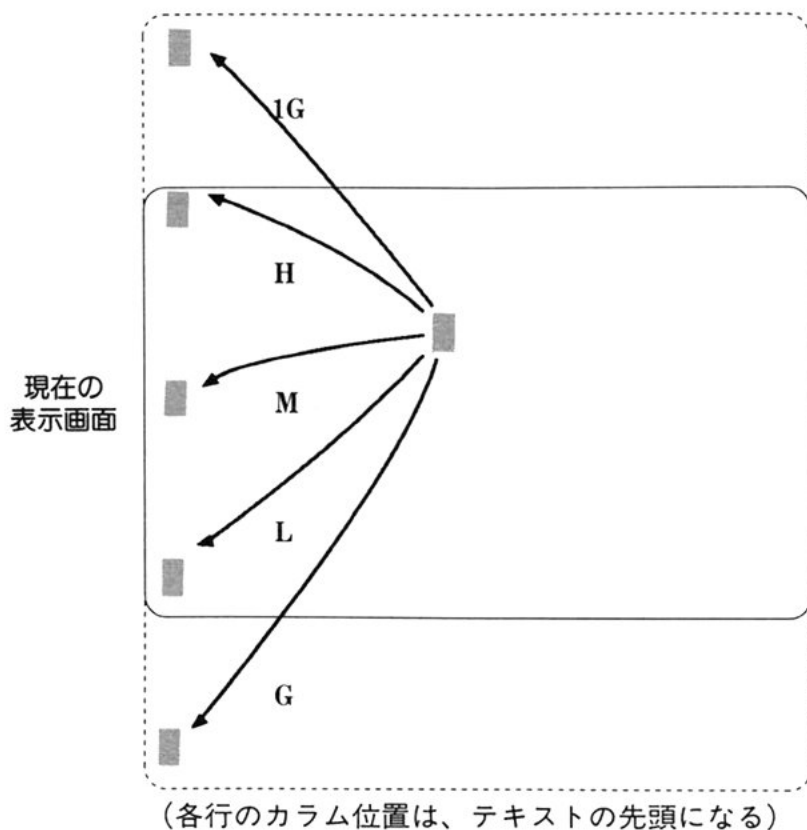
CTRL ＋b　1文字左へ移動（backward-char）
CTRL ＋f　1文字右へ移動（forward-char）
CTRL ＋p　1行上へ移動（previous-line）
CTRL ＋n　1行下へ移動（next-line）

■ **H**　現在の表示画面の左上端へ移動

■ **M**　現在の表示画面の中央の行へ移動

■ **L**　現在の表示画面の最下行へ移動

■ ***n*G**　ファイル全体の最初から *n* 行目へ移動
（*n* を指定しなければ最下行に移動）

（各行のカラム位置は、テキストの先頭になる）

Emacs

ESC ＋＜　バッファ（テキスト）の先頭に移動（beginning-of-buffer）

ESC ＋＞　バッファ（テキスト）の末尾に移動（end-of-buffer）

ESC ＋x goto-line ⏎ *n* ⏎　*n* 行目にカーソルを移動

カーソルの移動・検索

■ 0　現在の行の先頭へ移動

【例】

```
__________if (p>=129 & p<=159) {
          ↓ 0
__________if (p>=129 & p<=159) {
```

■ ⏎キー　次の行の左端へ移動

【例】

```
__________if (p>=129 & p<=159) {
__________________putchar(c);
          ↓ ⏎
__________if (p>=129 & p<=159) {
__________________putchar(c);
```

■ ^　空白を除いて現在の行の先頭へ移動

【例】

```
__________if (p>=129 & p<=159) {
          ↓ ^
__________if (p>=129 & p<=159) {
```

■ $　現在の行の末尾へ移動

【例】

```
__________if (p>=129 & p<=159) {
          ↓ $
__________if (p>=129 & p<=159) {
```

■ －　上の行の左端へ移動

【例】

```
----------if (p>=129 & p<=159) {
------------------putchar(c);
          ↓ -
----------if (p>=129 & p<=159) {
------------------putchar(c);
```

■ w　次の単語の先頭の文字へ移動

【例】

```
----------if (p>=129 & p<=159) {
          ↓ w
----------if (p>=129 & p<=159) {
```

■ b　1つ前の単語の先頭の文字へ移動

【例】

```
----------if (p>=129 & p<=159) {
          ↓ b
----------if (p>=129 & p<=159) {
```

■ e　現在または次の単語の末尾へ移動

【例】

```
----------if (p>=129 & p<=159) {
          ↓ e
----------if (p>=129 & p<=159) {
```

Emacs

CTRL ＋a	現在の行の先頭へ移動	(beginning-of-line)
CTRL ＋e	現在の行の末尾へ移動	(end-of-line)

カーソルの移動・検索

■ */str*⏎　指定した文字列（*str*）の先頭の文字へ移動（順方向に探す）

■ **f***chr*　行内の指定した１文字（*chr*）へ移動（順方向に探す）

【例】

```
if (p>=129 & p<=159) {
↓  /p⏎または fp ←―――― 指定した文字 p へ移動する
if (p>=129 & p<=159) {
↓  n            ←―――― 順方向に再検索する
if (p>=129 & p<=159) {
```

■ **?***str*⏎　指定した文字列（*str*）の先頭の文字へ移動（逆方向に探す）

■ **F***chr*　行内の指定した１文字（*chr*）へ移動（逆方向に探す）

【例】

```
if (p>=129 & p<=159) {
↓  ? p または Fp ←―――― 指定した文字 p へ移動する
if (p>=129 & p<=159) {
↓  N            ←―――― 逆方向に再検索する
if (p>=129 & p<=159) {
```

■ **n**　順方向に再検索

■ **N**　逆方向に再検索

Emacs

CTRL + s string	順方向にインクリメンタル検索（isearch-forward）
CTRL + r string	逆方向にインクリメンタル検索（isearch-backward）
CTRL + s ESC string ⏎	順方向に一括検索（search-forward）
CTRL + r ESC string ⏎	逆方向に一括検索（search-backward）
CTRL + s	順方向に再検索
CTRL + r	逆方向に再検索

4・3

編集（削除、置換、複写）

■ x　カーソルのある1文字を削除

【例】

```
putychar(c)
      ↓  x ←──── y を削除する
putchar(c)
```

■ X　カーソルの左の1文字を削除

【例】

```
putychar(c)
      ↓  X ←──── y を削除する
putchar(c)
```

■ dd　カーソルのある行を削除（他へ移動可能）

【例】

```
if (p>=129 & p<=159) {
        putchar(c);
} else {
      ↓  dd ←──── putchar (c); の行を削除する
if (p>=129 & p<=159) {
} else {
```

■ dw　カーソルのある位置からこの語の最後まで削除

【例】

```
#define DEL1 '¥010'
        ↓  dw ←―――――― この語の最後までを削除する
#define '¥010'
```

■ df *chr*　カーソルのある位置から指定した1文字(*chr*)までを削除

【例】

```
if (p>=129 && p<=159)  {
        ↓  df9 ←―――――― カーソルから9までの文字を削除する
if (p>=129)  {
```

■ d$　カーソルのある位置から右側を削除

【例】

```
#include <stdio.h>
        ↓  d$
#include
```

■ d^　カーソルのある位置から左側を削除

【例】

```
#include <stdio.h>
        ↓  d^
<stdio.h>
```

Emacs

DEL	カーソルの前の1文字を消去（delete-backward-char）
CTRL ＋d	カーソルの位置の1文字を消去（delete-char）
CTRL ＋k	カーソルの位置からその行の最後尾まで削除（kill-line）
CTRL ＋w	リージョンを削除し記憶（kill-region）
ESC ＋w	リージョンを削除せずに記憶（copy-region-as-kill）
CTRL ＋ｽﾍﾟｰｽ	カーソル位置にマーク（リージョン）
CTRL ＋@	カーソル位置にマーク（リージョン）
CTRL ＋x CTRL ＋x	マークとカーソルを入れ換え（リージョン）

■ r　カーソルのある1文字を次に指定した文字で置換

【例】

```
puychar(c)
      ↓  rt ←────── y を t で置換する
putchar(c)
```

■ R　カーソルのある位置から複数個の文字を置換

【例】

```
xxxchar(c) ;
      ↓  Rput ←────── xxx を put で置き換える
putchar(c) ;
```

■ s　カーソルのある1文字を複数個の文字で置換

【例】

```
xchar(c) ;
      ↓  sput ←────── x を put で置き換える
putchar(c) ;
```

■ S　カーソルのある行全体を置換

【例】

```
if (p>=129 & p<=159)  {
         putchar(c) ;
}  else  {
      ↓
if (p>=129 & p<=159)  {
         c=getchar( ) ;
}  else  {
```

S c=getchar( ) ;
putchar(c) ; を　c=getchar( ) ; に置き換える

■ **cc**　カーソルのある行全体を置換

■ **cw**　カーソルのある位置からこの語の最後まで置換

【例】

```
#define FALSE 0
      ↓ cw TRUE ESC
#define TRUE 0
```

■ **cf** *chr*　カーソルのある位置から指定した１文字(*chr*)まで置換

```
if (p>=129 & p<=159) {
      ↓ cf 9221 ←──── カーソルから9までを置換する
if (p>=129 & p<=221) {
```

Emacs

ESC +% *string1* ⏎ *string2* ⏎	確認をしながら *string1* を *string2* に置換
ESC +x replace-string *string1* ⏎ *string2* ⏎	*string1* を *string2* に一括置換

編集（削除、置換、複写）

■ .　修正を繰り返す

■ u　最後の修正を取り消す

■ U　この行に対して行ったすべての修正を取り消す

■ yw　コピーする単語を指定

■ yy　コピーする行を指定

■ p　カーソルのある行の下へ移動または複写

■ P　カーソルのある行の上へ移動または複写

【例】

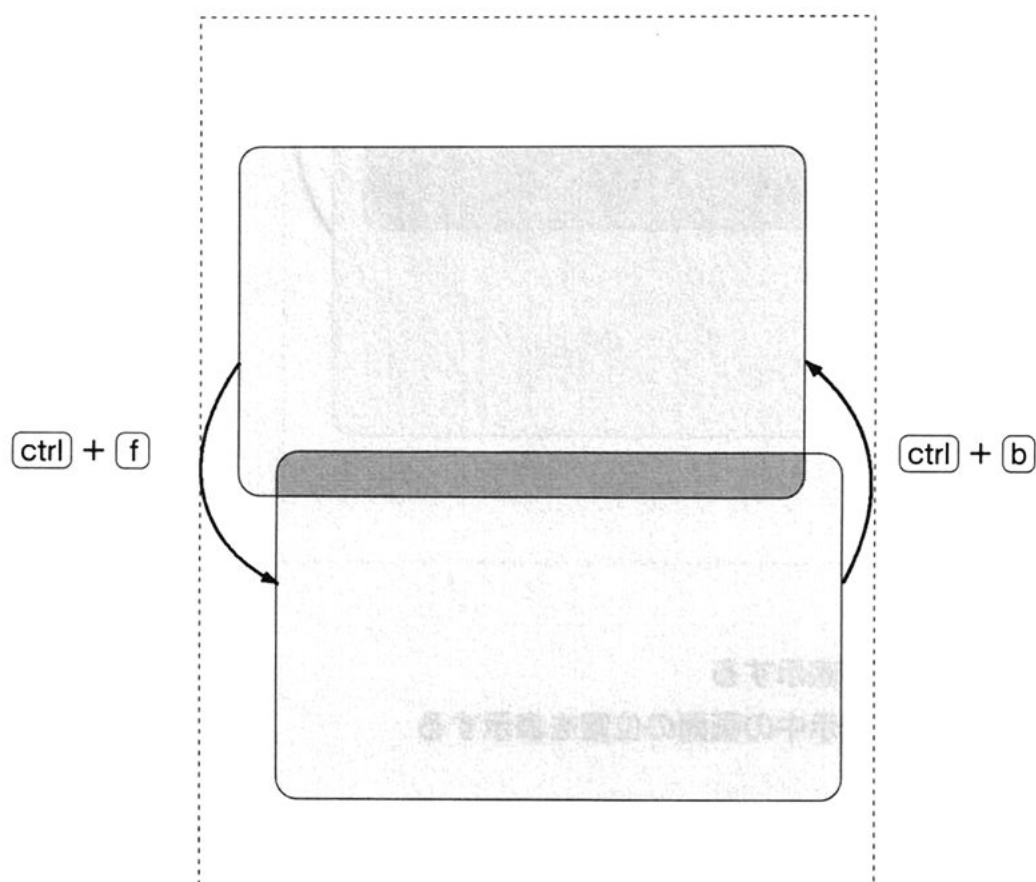
```
#include <stdio.h>
#define DEL1 '¥010'
       ↓ yyp ←――――コピーする行(カーソルの行)を指定し、
#include <stdio.h>      カーソル行の下に複写する
#include <stdio.h>
#define DEL1 '¥010'
```

Emacs

CTRL + y	最後に削除されたテキストを取り込む (yank)
CTRL + g	コマンドを中止 (keyboard-quit) (ミスタイプによってどうすれよいかわからなくなったとき)
CTRL + _	1回前の変更を取り消す (undo)

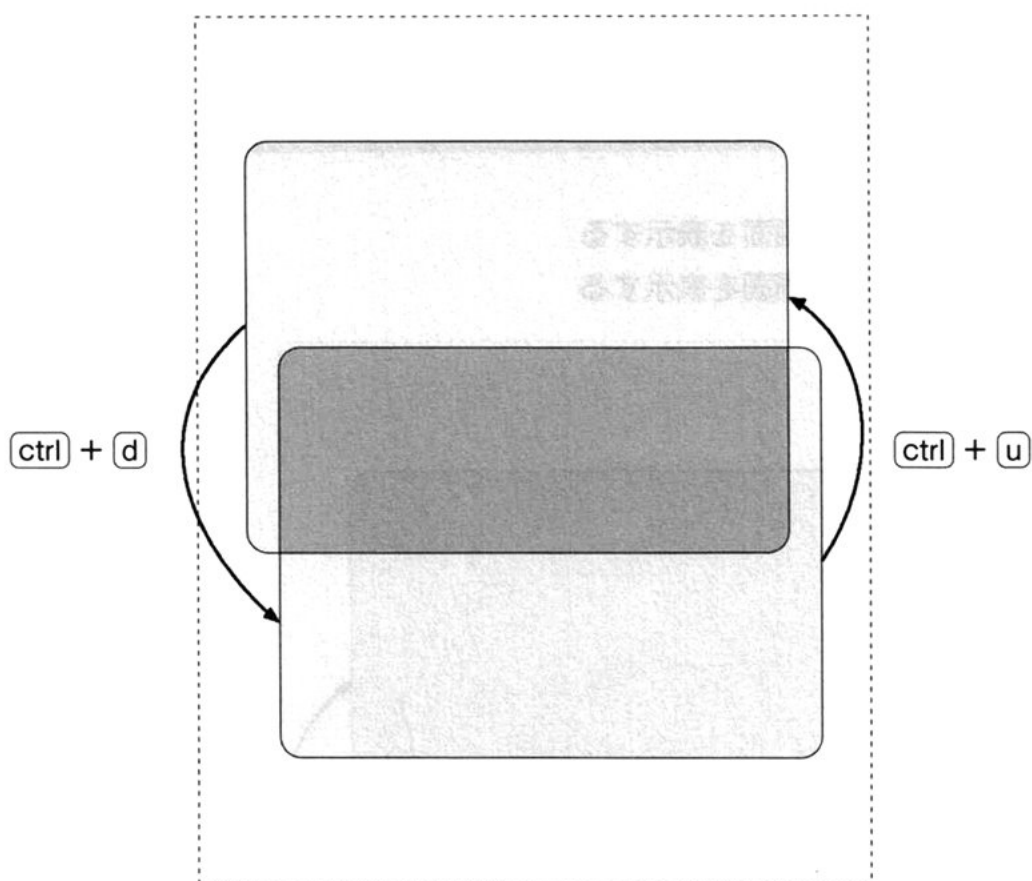
4・4

画面操作

■ ctrl ＋ f　　次の画面を表示する
■ ctrl ＋ b　　前の画面を表示する

Emacs

CTRL ＋ v	次の画面を表示（scroll-up）
ESC ＋ v	前の画面を表示（scroll-down）

画面操作

■ ctrl + d　半画面分次を表示する
■ ctrl + u　半画面分前を表示する

■ ctrl + l(r)　再表示する
■ ctrl + g　表示中の画面の位置を表示する

【例】

全 52 行のファイルの中で、26 行目にカーソルがあるときの表示例

"exfile.c" line 26 of 52 -50%-

ファイル名　全 52 行中の 26 行目

Emacs（ウィンドウ操作）

CTRL ＋x 2	ウィンドウを上下に分割する (split-window-vertically)
CTRL ＋x 5	ウィンドウを左右に分割する (split-window-horizontally)
CTRL ＋x 0	ウィンドウを削除する（delete-window)
CTRL ＋x 1	他のウィンドウをすべて削除する (delete-other-window)
CTRL ＋x ^	ウィンドウを上下に拡張する（enlarge-window)
CTRL ＋x }	ウィンドウを左右に拡張する (enlarge-window-horizontally)
CTRL ＋x o	別のウィンドウに移動する（other-window)

4・5

ファイル操作

■ **ZZ**　エディタを終了する
内容が変更されている時にはファイルに保存する。

■ **:w**[*file*]　指定したファイル(*file*)に保存する

■ **:wq**　ファイルに保存してエディタを終了する

■ **:q**　エディタを終了する
内容を変更した時には、警告メッセージが出力される。

■ **:q!**　エディタの強制終了
内容はファイルに保存されない。

■ **:e**[*file*]　別のファイル(*file*)を編集する

Emacs

CTRL +x	CTRL +s	バッファの内容をファイルに保存 (save-buffer)
CTRL +x	CTRL +w	指定したファイルに保存 (write-buffer)
CTRL +x	CTRL +f	ファイルを読み込む (find-file)
CTRL +x	CTRL +c	emacs を終了する (save-buffers-kill-emacs)

Emacs （バッファ操作）

CTRL +x b buffer	バッファ(buffer)を選択する (switch-to-buffer)
CTRL +x k	バッファを削除する（kill-buffer)
CTRL +x CTRL +b	バッファのリストを表示する（list-buffers)

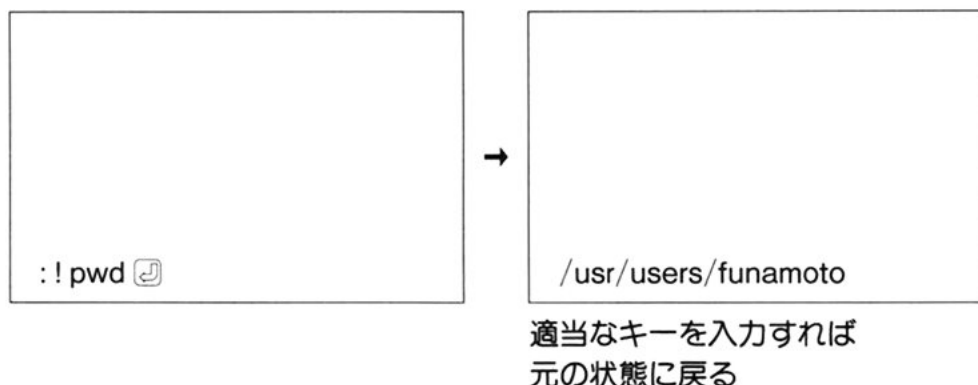
4・6

シェル

■ **:!** *cmd*　シェルコマンドを実行する

ワーキングディレクトリを確認する。

: ! pwd ⏎ → /usr/users/funamoto

適当なキーを入力すれば
元の状態に戻る

■ **!!** *cmd*　シェルコマンドを実行し、結果をカーソル位置に挿入する

: !! pwd ⏎ → /usr/users/funamoto

■ **:** *shell*　シェルへ制御を移す(shell は sh または csh を指定できる)

[CTRL]+[D]で元の状態に戻る。

Emacs

[ESC] + !	シェルコマンドの実行 (shell-command)
[ESC] +x *shell*	シェルの実行

シェル

4・7

その他の有用なコマンド

■ :set *number*　行番号を付ける

■ :set *nonumber* 行番号を消す

■ :r *file*　現在の行の次にファイル(*file*)を読み込む

【例】

```
%vi score1.dat ⏎              ←ファイルを指定してエディタを起動する
    m  susumu   90  80  70
    f  sumiyo   70  80  90
    m  tarou    40  20  50
   ↓ :r score2.dat ←カーソル行の次に score2. dat を読み込む
    m  susumu   90  80  70
    f  sumiyo   70  80  90
    m  tarou    40  20  50
    m  hiroshi  10   5  20 ←読み込まれた score2. dat
    f  hanako   60  70  80
```

その他の有用なコマンド

付録

付録A

UNIX 主要コマンド一覧

各社の UNIX でサポートされているコマンドのうち、ハードウェア依存性のあるものを除いたおおかたのコマンドをとりあげた。

コマンド名	機　能	対応	参照頁
adb	汎用デバッガ	SVR BSD	430
addbib	引用文献データベースの作成と拡張	BSD	
admin	SCCS ファイルの作成と更新	SVR BSD	
alias	コマンドに対する別名の割り当て	CSH	118
apply	指定した引数でコマンドを実行	BSD	
apropos	キーワードによるコマンドの検索	BSD	
ar	アーカイブファイルの作成と更新	SVR BSD	
as	アセンブラ	SVR BSD	
at	時刻指定のコマンド実行	SVR BSD	120
awk	パターン走査および簡易な処理をする言語	SVR BSD	122
banner	花文字の作成	SVR	129
basename	パス名からファイル名を抽出	SVR BSD	430
batch	バッチジョブの起動	SVR	130
bc	任意精度の計算用簡易言語	SVR BSD	131
bdiff	2つの大きなテキストファイルの差異を出力	SVR	
bfs	大きなファイルのパターン走査	SVR	
bg	バックグラウンド・ジョブの実行	CSH	134
biff	電子メールの着信通知の設定	BSD	135
binmail	電子メールの送信および受信	SVR BSD	

コマンド名	機　能	対応	参照頁
cal	カレンダーの出力	SVR BSD	136
calendar	スケジュールの管理	SVR BSD	138
cancel	ラインプリンタへの出力要求の取り消し	SVR	139
cat	ファイルの連結または出力	SVR BSD	140
cb	C 言語のプログラムソースの整形	SVR BSD	143
cc	C 言語コンパイラ	SVR BSD	145
ccat	圧縮したファイルの復元と出力	BSD	430
cd	ワーキングディレクトリの移動および変更	SVR BSD	147
cdc	SCCS デルタの注釈の変更	SVR BSD	
cflow	C 言語のフローチャートの出力	SVR	
checkeq	eqn ファイルの検査	BSD	
checknr	nroff, troff ファイルのチェック	BSD	
chfn	パスワードファイルのコメント内容の変更	BSD	
chgrp	ファイルのグループ ID の変更	SVR BSD	150
chmod	ファイルの許可モードの変更	SVR BSD	151
chown	ファイルの所有者の変更	SVR BSD	153
chsh	ログイン時に起動するシェルの変更	BSD	155
clear	画面のクリア	BSD	157
cmp	2 つのファイルの内容比較	SVR BSD	158
col	逆改行の処理	BSD	
colcrt	nroff 画面出力時の改行制御	BSD	
colrm	指定カラムの削除	BSD	161
comb	SCCS ファイル整理用のシェルスクリプトの作成	SVR BSD	
comm	繰り返し行の消去および出力	SVR BSD	162
compress	ファイルの圧縮	SVR BSD	431
convert	アーカイブファイルを共通形式に変換	SVR	

コマンド名	機　能	対応	参照頁
cp	ファイルのコピー	SVR BSD	164
cpio	アーカイブファイルのコピー	SVR	
cpp	C 言語コンパイラのプリプロセッサ	SVR BSD	
crontab	時刻指定のコマンド実行	SVR	
crypt	ファイルの暗号化	BSD	
csh	C シェルの起動	BSD	166
csplit	文脈によるファイルの分割	SVR	
ctags	タグファイルの作成	BSD	
ct	リモート端末に対する getty の生成	SVR	
cu	他の UNIX システムの呼び出し	SVR BSD	431
cut	ファイルのフィールド単位の切り出し	SVR	168
cxref	C 言語のクロスリファレンスリストの出力	SVR	431
date	日付の出力と設定	SVR BSD	170
dbx	プログラムのソースレベルのデバッガ	BSD	172
dc	卓上計算用プログラム	SVR BSD	432
dd	ファイルのフォーマット変換およびコピー	SVR BSD	176
delta	SCCS ファイルへのデルタの登録	SVR	
deroff	nroff, troff 構文の除去	SVR BSD	177
df	ファイルシステムに関する情報の出力	SVR BSD	180
diction	文および文節の数の出力	BSD	
diff	2 つのテキストファイルの差異を出力	SVR BSD	182
diff3	3 つのテキストファイルの差異を出力	SVR BSD	432
dircmp	ディレクトリの比較	SVR	185
dirname	パス名からのディレクトリ名の取り出し	SVR	432
dirs	ディレクトリスタックの内容の出力	CSH	187
disable	ラインプリンタへの出力不能を設定	SVR	432
du	ディレクトリの使用状況の出力	SVR BSD	188

コマンド名	機　能	対応	参照頁
echo	引数の内容の出力	CSH	190
ed	テキストエディタ	SVR BSD	191
edit	テキストエディタ（ex の簡易版）	SVR BSD	433
efl	拡張 FORTRAN 言語コンパイラ	BSD	
egrep	ファイル中の指定した文字列やパターンを走査して出力	SVR BSD	204
enable	ラインプリンタへの出力可能を設定	SVR	433
enroll	シークレットメールの送信と受信	BSD	
env	コマンド実行時の環境設定	SVR	
eqn	数式の清書	BSD	
error	コンパイラからのエラーメッセージをテキストに挿入	BSD	433
ex	ed の上位互換のテキストエディタ	SVR BSD	194
expand	タブをスペースに展開	BSD	433
explain	対話形式での diction 処理	BSD	
expr	引数を式として計算	SVR BSD	
eyacc	拡張版 yacc	BSD	
f77	FORTRAN77 言語コンパイラ	BSD	
factor	素因数分解	SVR	
false	常に偽	SVR BSD	
fed	フォントファイルの編集	BSD	
fg	フォアグラウンド・ジョブとしての実行	CSH	196
fgrep	ファイル中の指定した文字列やパターンを走査して出力	SVR BSD	204
file	ファイルの種類の検査と出力	SVR BSD	197
find	ファイルの存在する位置の探査と出力	SVR BSD	198
finger	ユーザの情報の出力	BSD	201

コマンド名	機　能	対応	参照頁
fmt	簡易テキストフォーマッタ	BSD	
fold	1行当たりの出力幅の設定	BSD	203
fp	関数型のプログラム言語	BSD	
fpr	FORTRAN 言語用の改行変換	BSD	
from	メールの発信者および発信日付の出力	BSD	434
fsplit	FORTRAN 言語ソースの分割	BSD	
ftp	ネットワーク上でのファイル転送	SVR BSD	434
gcore	走行プロセスのコアダンプの出力	BSD	
gdev	グラフィックス用のフィルタ	SVR	
ged	グラフィックス用のエディタ	SVR	
get	SCCS ファイルからの特定バージョンの回復	SVR BSD	
getopt	コマンドオプションの構文解析	SVR	
glossary	一般的な UNIX の用語と記号の定義と出力	SVR	
gprof	プロファイルの出力	BSD	434
graph	グラフの描画	SVR BSD	
graphics	グラフィックスおよび数値処理用コマンドへのアクセス	SVR	
greek	端末フィルタの選択	SVR	
grep	ファイル中の指定した文字列やパターンを走査して出力	SVR BSD	204
groups	所属するグループ名の出力	SVR BSD	209
gutil	グラフィックス用ユーティリティ	SVR	
hashcheck	スペルリストの再生成	SVR	
hashmake	スペルリストの生成	SVR	
head	ファイルの先頭部分の出力	SVR BSD	210
help	メッセージ番号および SCCS コマンドのヘルプ機能	SVR	

コマンド名	機　能	対応	参照頁
help	簡単なUNIXコマンドの解説	BSD	
helpadm	ヘルプデータベースの更新	SVR	
history	コマンド履歴の出力	CSH	211
hostid	ホスト識別番号の出力	BSD	214
hostname	使用中のホスト名の出力	SVR BSD	215
id	ユーザ名、ユーザID、グループ名、グループIDの出力	SVR	216
indent	C言語プログラムソースの書式整備	BSD	217
indxbib	文献目録インデックスの作成	BSD	
install	バイナリファイルのインストール	SVR BSD	
iostat	システムの入出力に関する統計情報の出力	BSD	
ipcrm	メッセージキュー、セマフォセット、共有メモリ識別子の削除	SVR	
ipcs	プロセス間通信の状態の出力	SVR	
jobs	ジョブの状況の出力	CSH	219
join	2つのファイルの関係づけた結合	SVR BSD	221
kill	ジョブの強制終了	CSH	223
last	ログインおよびログアウトの情報を最新のものから出力	BSD	225
lastcomm	コマンド使用状況の情報を最新のものから出力	BSD	
ld	リンケージエディタ	SVR BSD	
learn	コマンド学習用のCAI	BSD	
leave	端末を離れる時刻の通知設定	BSD	227
lex	字句解析プログラムの生成	SVR BSD	434
line	1行の読み込み	SVR	435
lint	C言語ソースプログラムの文法チェック	SVR BSD	228
lisp	LISP言語インタプリンタ	BSD	

コマンド名	機　能	対応	参照頁
liszt	LISP言語コンパイラ	BSD	
ln	ファイルのハードおよびシンボリックリンク	SVR BSD	231
locate	キーワードによるUNIXコマンドの検索	SVR	435
lock	端末のロック	BSD	233
login	ログイン(セッション開始)	SVR BSD	234
logname	ログイン名の出力	SVR	235
logout	ログアウト(セッション終了)	CSH	236
look	ソートされたファイルから1行の出力	BSD	
lookbib	目録の検索	BSD	
lorder	オブジェクトファイルの参照関係の出力	SVR BSD	435
lp	ラインプリンタへの出力要求	SVR	237
lpq	スプールにあるプリントジョブの状況の出力	BSD	239
lpr	プリンタ(スプール)へのジョブの出力	BSD	240
lprm	スプールにあるプリントジョブの削除	BSD	242
lpstat	プリンタの状況の出力	SVR	243
ls	ディレクトリの内容の出力	SVR BSD	245
lxref	LISP言語のクロスリファレンスリストの出力	BSD	
m4	マイクロプロセッサ	SVR BSD	
machid	プロセッサタイプの判定出力	SVR	
mail	電子メールの発信および受信	SVR BSD	249
mailx	会話型のメッセージ処理	SVR	
make	プログラムやファイルの関連づけた保守および更新	SVR BSD	252
makekey	暗号キーの生成	SVR	
man	オンライン・リファレンス・マニュアル	SVR BSD	255

コマンド名	機　能	対応	参照頁
mesg	メッセージ受け付けの可否の設定	SVR BSD	259
mkdir	ディレクトリの作成	SVR BSD	260
mkstr	エラーメッセージファイルの作成	BSD	
more	テキストファイルの1ページ毎の出力	SVR BSD	261
msgs	システムメッセージの出力	BSD	
mt	MT 装置の操作	BSD	
mv	ファイルの移動および名前変更	SVR BSD	264
neqn	端末への数式の出力	BSD	
netstat	ネットワークの状態の出力	BSD	435
newaliases	メール・エリアス・データベースの新規作成	BSD	
newform	テキストファイルの形式変更	SVR	
newgrp	グループ ID の変更	SVR	436
news	ニュースの出力	SVR	436
nice	指定した優先順位でのコマンドの実行	SVR BSD CSH	266
nl	行番号付けのファイルの出力	SVR	436
nm	オブジェクトファイルのネームリストの出力	SVR BSD	
nohup	ハングアップやクイットを無視したコマンドの実行	SVR BSD	437
nroff	テキスト清書処理	SVR BSD	267
od	ファイルのダンプ	SVR BSD	269
pack	ファイルの圧縮	SVR	437
page	テキストファイルの1ページ毎の出力	BSD	261
pagesize	システムのページサイズの出力	BSD	
passwd	パスワードの設定または変更	SVR BSD	272
paste	複数ファイルの同一行のマージ	SVR	273

コマンド名	機　能	対応	参照頁
pc	pascal 言語コンパイラ	BSD	
pcat	圧縮したファイルの内容出力	SVR	437
pdx	pascal 言語デバッガ	BSD	
pg	テキストファイルの1ページ毎の出力	SVR	274
pi	pascal 言語のPコード・トランスレータ	BSD	
pix	pascal 言語インタプリンタ	BSD	
plot	グラフ出力(プロッタ)用のフィルタ	BSD	437
pmerge	pascal 言語ファイルのマージ	BSD	
popd	スタックからのディレクトリのポップアップ	CSH	276
pr	プリンタ印刷用のフィルタ	SVR BSD	277
print	ラインプリンタに対する出力要求	BSD	437
printenv	環境変数の値の出力	BSD	278
prmail	電子メールの読み出し	BSD	
prof	プロファイルデータの出力	SVR BSD	
prs	SCCS ファイルの出力	SVR	
ps	現在のプロセスの状態の出力	SVR BSD	280
pti	写植機用インタプリタ	BSD	
ptx	キーワードによるインデックス出力	BSD	
pushd	ディレクトリのスタックへの格納	CSH	283
pwd	ワーキングディレクトリのパス名の出力	SVR BSD	284
px	pascal 言語インタプリタ	BSD	
pxp	pascal 言語実行プロファイラ	BSD	
pxref	pascal 言語クロスリファレンス	BSD	
ranlib	アーカイブファイルのランダムファイルへの変換	BSD	
ratfor	構造化 FORTRAN 言語	BSD	

コマンド名	機　能	対応	参照頁
rcp	リモートシステム間でのファイルのコピー	SVR BSD	285
refer	nroff および troff のプリプロセッサ	BSD	
reset	端末オプションのリセット	BSD	438
rev	テキストファイルの1行を逆順に出力	BSD	438
rlogin	リモートシステムへのログイン	SVR BSD	287
rm	ファイルの削除	SVR BSD	288
rmail	リモート電子メール	SVR BSD	438
rmdel	SCCS デルタの削除	SVR BSD	
rmdir	ディレクトリの削除	SVR BSD	289
rsh	リモートシステムでのシェルの起動	BSD	230
rsh	限定機能シェル	SVR	290
ruptime	ローカルネットワーク上のマシンの稼働状況報告	SVR BSD	291
rwho	ローカルネットワーク上のユーザ状況報告	SVR BSD	293
sact	SCCS ファイルの編集作業状況の出力	SVR BSD	
sccsdiff	SCCS ファイルの2つのファイルの内容比較	SVR BSD	
script	端末上でのセッションの記録	BSD	295
sdb	シンボリックデバッガ	SVR BSD	297
sdiff	2つのファイルの行毎の相違比較	SVR	
sed	ストリーム(非会話型)エディタ	SVR BSD	298
sendbug	バグレポートの出力	BSD	
set	シェル変数値の出力または設定	CSH	300
setenv	環境変数値の出力または設定	CSH	303
sh	標準シェルの起動	SVR BSD	304
size	実行形式ファイルの各セクションの大きさの出力	SVR BSD	305
sleep	コマンド実行の一時中断	SVR BSD	306

コマンド名	機　能	対応	参照頁
sort	ファイルのソートまたはマージ	SVR BSD	307
sortbib	目録のソート	BSD	
source	環境設定ファイルの内容のコマンド実行	CSH	310
spell	英文スペルチェッカ	SVR BSD	
spellin	スペルリストへの追加	SVR BSD	
spellout	スペルリストからの削除	BSD	
spline	スプライン曲線の補間座標の計算	BSD	311
split	ファイルの分割	SVR BSD	313
stat	グラフィックス用統計ネットワーク	SVR	
stop	ジョブの停止	CSH	315
strings	オブジェクトファイル中の表示可能文字の出力	BSD	
strip	オブジェクトファイル中のシンボル表またはリロケーション情報の削除	SVR BSD	
struct	構造化 FORTRAN 言語	BSD	
stty	端末オプションの出力または設定	SVR BSD	316
style	文書中の文体の解析	BSD	
su	一時的なスーパーユーザまたは新ユーザへの切り替え	SVR BSD	318
sum	ファイルのチェックサムおよびブロック数の出力	SVR BSD	438
symorder	ネームリストの再構成	BSD	
sync	スーパーブロックの更新	SVR BSD	
tabs	端末のタブ設定	SVR BSD	438
tail	ファイルの末尾部分の出力	SVR BSD	319
talk	他のユーザとの会話	SVR BSD	321
tar	ファイルのテープ保存またはテープからのファイルの復旧	SVR BSD	323

コマンド名	機　能	対応	参照頁
tbl	nroff および troff 用の表作成	SVR BSD	
tc	写植機シミュレータ	BSD	
tee	パイプによる接続	SVR BSD	325
telnet	TELNET プロトコルによるリモートシステムとの通信	BSD	438
test	条件判定	SVR BSD	326
time	コマンドの実行時間の出力	SVR BSD	328
timex	コマンドの実行時間、プロセスデータ、システムアクティビティの出力	SVR	439
tip	他の UNIX システムとの接続	BSD	439
touch	ファイルの最終更新日時の変更	SVR BSD	439
tp	テープアーカイバの処理	BSD	
tplot	グラフィックス用のフィルタ	SVR	439
tput	terminfo データベースの照会	SVR	
tr	文字列の変換	SVR BSD	329
troff	写植機用の文書処理	SVR BSD	330
true	常に真	SVR BSD	
tset	端末制御キーの設定	BSD	332
tsort	トポロジカルソート	SVR BSD	440
tty	端末名の出力	SVR BSD	333
ul	アンダーライン付与	BSD	
umask	ファイル作成時の保護モードの設定	SVR	440
unalias	コマンドの別名設定の解除	CSH	334
uname	UNIX システムの名称出力	SVR	335
uncompress	圧縮ファイルの復元	SVR BSD	440
unexpand	スペースをタブに置換する	BSD	440

コマンド名	機　能	対応	参照頁
unget	直前の SCCS ファイルに対する get の取り消し	SVR BSD	
uniq	ファイル内の重複行の出力	SVR BSD	336
units	単位変換	SVR BSD	441
unpack	圧縮ファイルの復元	SVR	441
unset	シェル変数値の設定の解除	CSH	338
unsetenv	環境変数値の設定の解除	CSH	340
uptime	システムが起動された時間、稼働時間、ユーザ数、ロードアベレージの出力	BSD	341
usage	コマンドの解説と使用法の出力	SVR	441
users	ログイン中のユーザ名の出力	BSD	342
uucp	UNIX システム間のファイルの転送	SVR BSD	441
uudecode	ファイルの複号化	BSD	
uuencode	ファイルの暗号化、復合化	BSD	
uulog	uucp と uux の状況出力	SVR BSD	441
uuname	他の UNIX システムの uucp 名の出力	SVR	442
uupick	転送されたファイルの許可または不許可	SVR	
uusend	リモートシステムへのファイルの転送	BSD	442
uustat	uucp の状況出力	SVR	442
uuto	他の UNIX システムへのファイルの転送	SVR	442
uux	他の UNIX システムでのコマンドの実行	SVR BSD	443
val	SCCS ファイルの検索	SVR	
vc	バージョン管理	SVR	
vfontinfo	フォント情報の出力	BSD	
vgrind	プログラムソースリストの troff 用変換	BSD	
vi	vi エディタ（フルスクリーンエディタ）の起動	SVR BSD	343

コマンド名	機　能	対応	参照頁
view	vi エディタの書き込み禁止モードでの起動	SVR BSD	345
vlp	LISP 言語のプログラムソースリストの nroff,troff 用変換	BSD	
vmstat	仮想メモリの状態出力	BSD	
w	現在ログイン中のユーザの実行コマンドの出力	BSD	346
wait	プロセスの終了待ち	SVR BSD	443
wall	全ユーザへのメッセージの送信	SVR BSD	348
wc	ファイル中の文字数、単語数、行数の出力	SVR BSD	349
what	SCCS ファイル内の文字の識別 オブジェクトファイルのバージョン出力	SVR BSD	
whatis	コマンドの簡単な説明の出力	BSD	350
whereis	プログラムのソース、バイナリ、マニュアルの各ファイルの存在場所の出力	BSD	351
which	コマンドの所在の出力	BSD	352
who	現在ログイン中のユーザ名の出力	SVR BSD	353
whoami	現在のログイン名を出力	BSD	
write	特定のユーザへのメッセージの送信	SVR BSD	355
xargs	引数リストの作成およびコマンドの実行	SVR	
xget	秘密メールの受信	BSD	
xsend	秘密メールの発信	BSD	
xstr	C 言語プログラム中の共有文字列の出力	BSD	443
yacc	コンパイラコンパイラ	SVR BSD	443
yes	文字列の繰り返し出力	BSD	443

付録B

UNIXコマンド第2レベル

コマンド・リファレンスでとりあげた約140コマンドに続いて重要なコマンドを説明した。

adb　　アセンブラデバッガ　　BSD

【書式】 adb［*options*］［*objfile*［*corefile*］］

【解説】 実行形式ファイル（objfile）をコアイメージファイル（*corefile*）を使って会話形式でデバッグする。

　-w　：*objfile* と *corefile* を作成する。

basename　　パス名からのファイル名抽出　　SVR BSD

【書式】 basename *str*［*suffix*］

【解説】 指定した文字列（*str*）から（/）までのプレフィックスとサフィックスを取り除いたものを出力する。

ccat　　圧縮したファイルの復元と出力　　

【書式】 ccat［*files*］

【解説】 圧縮されたファイル（*files*）を復元する。

compress　ファイルの圧縮　SVR BSD

【書式】 compress [*options*] [*files*]

【解説】 ファイル（*files*）を圧縮し、file.Zに格納する。

-b*n*　：コード化のビット数（*n*）を制限する。
　　　省略時：*n*＝16

-c　：標準出力に書き出す。

-f　：強制的に圧縮する。

-v　：圧縮比率を出力する。

cu　他のUNIXシステムの呼び出し　SVR BSD

【書式】 cu [*options*] [*telno*] [*system*]

【解説】 指定した電話番号（*telno*）またはシステム名（*system*）で別のUNIXを呼び出す。

-s*speed*　：伝送速度（*speed*）を指定する。

-l*line*　：通信回線装置名（*line*）を指定する。

-h　：二重モードのシステムを対象とする。

-t　：自動応答モードになっているASCII端末を対象とする。

-d　：診断メッセージを出力する。

-o　：奇数パリティの生成を指示する。

-e　：偶数パリティの生成を指示する。

-n　：電話番号の入力を要求する。

cxref　C言語のクロスリファレンスリストの出力

【書式】 cxref [*options*] *files*

【解説】 C言語のソースリスト(*files*)のクロスリファレンスリストを作成する。

-c　：入力ファイルすべてを対象としたリストを出力する。

-w*num*　：出力の1行の幅（*num*）を指定する。

-o *file*　：ファイル（*file*）に出力する。

-s　　：入力ファイル名を出力しない。
-t　　：1行80文字で出力する。

dc　　卓上計算用プログラム　　SVR BSD

【書式】 dc［*file*］
【解説】 ファイル（*file*）の内容を読み込み、高精度の計算を行う。

diff3　　3つのテキストファイルの差異を出力　　SVR BSD

【書式】 diff3［*options*］*file1 file2 file3*
【解説】 指定した3つのファイル（*file1*, *file2*, *file3*）の違いを出力する。
-e　：相違結果をedコマンドのスクリプトとして出力する。
-x　：file 1に相違結果を組み込むためのedコマンドのスクリプトを出力する。

dirname　　パス名からのディレクトリ名の取り出し　　SVR

【書式】 dirname *string*
【解説】 パス名（*string*）からファイル名の部分を除く絶対修飾のディレクトリ名を取り出す。

disable　　ラインプリンタへの出力不能を設定　　SVR

【書式】 disable［*option*］*printers*
【解説】 プリンタ（*printers*）にlpコマンドによる出力要求を受け付けないように設定する。
-c　　：印刷中の要求を取り消す。
-r［*reason*］：作動停止の理由（*reason*）を付ける。

edit　テキストエディタ（ex の簡易版）　SVR BSD

【書式】 edit ［*option*］ *files*

【解説】 テキストファイル（*files*）を編集する。

-r　：障害時にファイルの復旧をする。

enable　ラインプリンタへの出力可能を設定　SVR

【書式】 enable *printers*

【解説】 プリンタ(*printers*)に対して lp コマンドからの出力要求を受け付けるように設定する。

error　コンパイラからのエラーメッセージをテキストに挿入　BSD

【書式】 error ［*options*］ ［*file*］

【解説】 ファイル（*file*）または標準入力からコンパイラの出力を読み込み、エラーメッセージをソースファイルに挿入する。

-I*file*　：無視する関数名のファイル（*file*）を指定する。

-n　：標準出力に書き出す。

-q　：書き込みの際に確認を行う。

-v　：error コマンドの実行後 vi コマンドを起動する。

expand　タブをスペースに展開　BSD

【書式】 expand ［-*n*］ ［*files*］

【解説】 ファイル（*files*）中のタブを指定個数（n）の空白に置き換える。n を省略すれば $n=8$。

from　　メールの発信者および発信日付の出力　BSD

【書式】 from [*option*] [*name*]

【解説】 メールの発信者を出力する。ユーザ名(*name*)を指定すれば、そのユーザのメールのヘッダを出力する。

-s*sname*　：発信者（*sname*）だけのメールを対象とする。

ftp　　ネットワーク上でのファイル転送　SVR BSD

【書式】 ftp [*option*] [*hostname*]

【解説】 相手のホスト（*hostname*）との間でファイル転送を行う。

-d　：デバッグモード

-i　：プロンプトを表示しない。

gprof　　プロファイルの出力　BSD

【書式】 gprof [*options*] [*objfile* [*profile*]]

【解説】 プログラム(*objfile*)実行時の関数呼び出し回数、消費時間を記録したプロファイル(*profile*)を出力する。*objfile* 省略時は a.out、*profile* 省略時は gmon.out が対象となる。cc コマンドで-pg オプションが指定されている必要あり。

-c　：静的な呼出し関係を表示する。

-e *name*　：指定したルーチン名（*name*）のエントリを表示しない。

-f *name*　：指定したルーチン名（*name*）のエントリだけ表示する。

-z　：利用されなかったルーチンも表示する。

lex　　字句解析プログラムの生成　SVR BSD

【書式】 lex [*options*] [*files*]

【解説】 ファイル(*files*)に記述されたC言語および正規表現から、字句解析プログラムを生成する。

-t　：結果を標準出力に書き出す。
-v　：統計情報を要約する。
-n　：全統計情報を出力する。
-f　：高速コンパイル実行

line　1行の読み込み　SVR

【書式】 line
【解説】 標準入力から文字を読み込み、標準出力へ書き出す。

locate　キーワードによるUNIXコマンドの検索　SVR

【書式】 locate ［*keywords*］
【解説】 キーワード（*keywords*）を指定してコマンドを検索する。

lorder　オブジェクトファイルの参照関係の出力　SVR BSD

【書式】 lorder *files*
【解説】 オブジェクトファイル（*files*）の参照関係を出力する。

netstat　ネットワークの状態の出力　BSD

【書式】 netstat ［*option*］［*int*］
【解説】 ネットワークの状態を出力する。*int*（秒）を指定すればこの間隔で繰り返しレポートする。
-a　：全ソケットの状態を出力する。
-r　：ルーティングテーブルを出力する。
-s　：プロトコルごとの統計情報を出力する。

newgrp　　グループIDの変更　　SVR

【書式】 newgrp ［-］［*group*］

【解説】 指定したグループ（*group*）でログインする。

- ：新しいグループで再ログインしたのと同じ環境に設定される。

news　　ニュースの出力　　SVR

【書式】 news ［*options*］［*items*］

【解説】 /usr/news にあるファイルに記述された内容を、項目（*items*）を指定してユーザに知らせる。

-a　：通用時刻にかかわらず、すべての項目を出力する。

-n　：項目の名前だけを出力する。

-s　：項目の数だけを出力する。

nl　　行番号付けでのファイルの出力　　SVR

【書式】 nl ［*options*］ *file*

【解説】 ファイル（*file*）の内容に行番号を付けて出力する。

-b*type*　：番号付けをする行の種類（*type*）を指定する。

type＝a：すべての行

t：印字可能な行

n：番号付けなし

p*exp*：指定した正規表現（*exp*）の行

-p　：ページ毎に行番号をリセットしない。

-s*sep*　：行番号とテキストを区切る文字（*sep*）を指定する。

-n*format*　：行番号の書式（*format*）を指定する。

format＝ln：先行する0を削除し、左詰め

rn：先行する0を削除し、右詰め

rz：左詰め

nohup　　ハングアップやクイットを無視したコマンドの実行

【書式】 nohup *cmd* ［*args*］

【解説】 ハングアップやクイットを無視してコマンド（*cmd* ［*args*］）を実行する。

pack　　ファイルの圧縮 SVR

【書式】 pack ［*options*］ *files*

【解説】 ファイル（*files*）を圧縮する。

- ：各バイトの使用回数、相対頻度、バイトコードを出力する。

-f ：強制圧縮する。

pcat　　圧縮したファイルの内容出力 SVR

【書式】 pcat *files*

【解説】 pack コマンドで圧縮されたファイル（*files*）を復元する。

plot　　グラフ出力（プロッタ）用のフィルタ BSD

【書式】 plot ［*options*］ ［*files*］

【解説】 ファイル（*files*）または標準入力からコマンドを読み込み、グラフをプロットする命令を作成する。

-T *term* ：ターミナルの型（*term*）を指定する。

-r*res* ：解像度（*res*）を指定する。

print　　ラインプリンタに対する出力要求 BSD

【書式】 print ［*files*］

【解説】 指定したファイル（*files*）を pr コマンドでフォーマットし、lpr コマンドでプリンタ出力する。

reset　　端末オプションのリセット

【書式】 reset

【解説】 stty コマンドで設定したオプションを元に戻す。

rev　　テキストファイルの1行を逆順に出力

【書式】 rev ［*files*］

【解説】 入力したファイル（*files*）の各行の文字を逆順にして出力する。

rmail　　リモート電子メール

SVR BSD

【書式】 rmail ［*users*］

【解説】 uucp コマンド用の電子メールユーザ（*users*）を送る。

sum　　ファイルのチェックサムおよびブロック数の出力

SVR BSD

【書式】 sum *file*

【解説】 指定したファイル（*files*）のチェックサムとブロック数を出力する。

tabs　　端末のタブ設定

【書式】 tabs ［*option*］ *term*

【解説】 指定したターミナル（*term*）のタブを設定する。

-n　：左マージンの字下げをしない。

telnet　　TELNET プロトコルによるリモートシステムとの通信

BSD

【書式】 telnet ［*host* ［*port*］］

【解説】 telnet プロトコルで別のホスト（*host* ［*port*］）と通信する。

timex　コマンドの実行時間、プロセスデータ、システムアクティビティの出力　SVR

【書式】 timex ［*options*］ *cmd*

【解説】 コマンド（*cmd*）の実行時間、その他の各種データを計測する。

-p　：コマンドとその子プロセスの計測データを出力する。

-o　：コマンドとその子プロセスの読み込み／書き込みブロック数と転送文字数を出力する。

-s　：コマンドの実行中のシステムアクティビティ数を出力する。

tip　他の UNIX システムとの接続

【書式】 tip ［*options*］［*name*］［*number*］

【解説】 指定したシステム名(*name*)、または電話番号(number)で、別の UNIX と接続する。

-v　：tip コマンド変数の設定状況を出力する。

-speed　：指定したボーレート（speed）接続する。

touch　ファイルの最終更新日時の変更

【書式】 touch ［*options*］ *files*

【解説】 ファイル（*files*）の最終更新日時を変更する。

-c　：存在しないファイルは作成しない。

-f　：読み出し／書き込みの許可にかかわらず、変更を強行する。

tplot　グラフィックス用のフィルタ

【書式】 tplot ［*options*］

【解説】 プロッタ用のグラフィックスデータに変換する。

-T *term*　：ターミナルの型（*term*）を指定する。

-e *file*　：走査変換したファイル（*file*）をプロッタに送る。

tsort　トポロジカルソート　SVR BSD

【書式】 tsort ［*file*］

【解説】 指定したファイル（*file*）の内容をトポロジカルソートする。

tty　端末名の出力　SVR BSD

【書式】 tty ［*option*］

【解説】 使用中の端末のデバイス名を出力する。

-s　：端末＝0、非端末＝1を出力する。

umask　ファイル作成時の保護モードの設定　SVR

【書式】 umask ［*ooo*］

【解説】 ファイルの保護モードのマスク（*ooo*：8進数3文字）を指定する。

uncompress　圧縮ファイルの復元　SVR BSD

【書式】 uncompress ［*options*］［*files*］

【解説】 compressコマンドで圧縮したファイル（*files*）を復元する。

-c　：標準出力に書き出す。

-f　：無条件に復元する。

-v　：圧縮率を出力する。

unexpand　スペースをタブに置換する　BSD

【書式】 unexpand ［*option*］［*files*］

【解説】 指定したファイル（*files*）の空白をタブにする。

-a　：2個以上の空白があればタブにする。

units　単位変換　SVR BSD

【書式】 units

【解説】 数の単位を対話形式で変換する。

unpack　圧縮ファイルの復元　SVR

【書式】 unpack *files*

【解説】 pack コマンドで圧縮されたファイル（*files*）を復元する。

usage　コマンドの解説と使用法の出力　SVR

【書式】 usage［*options*］［*cmd*］

【解説】 指定したコマンド（*cmd*）の説明や使用例を検索して出力する。

-d　：機能説明を出力する。

-e　：使用例を出力する。

-o　：オプションの説明をする。

uucp　UNIX システム間のファイルの転送　SVR BSD

【書式】 uucp［*options*］*sfiles dfiles*

【解説】 UNIX システム間でファイル（*sfiles* → *dfiles*）を複写する。

-c　：スプールディレクトリに複写しない。

-d　：複写先にディレクトリがないときは作成する。

-m　：複写後、完了通知のメールが発信される。

uulog　uucp と uux の状況出力　SVR BSD

【書式】 uulog［*options*］

【解説】 /usr/spool/uucp/LOGFILE に格納された uucp, uux の情報を要約して出力する。

-s*sys*　：指定したシステム（*sys*）に関する情報のみ出力する。
-u*user*　：指定したユーザ（*user*）に関する情報のみ出力する。

uuname　他の UNIX システムの uucp 名の出力 SVR

【書式】 uuname ［*option*］
【解説】 uucp コマンドが認識している UNIX のシステム名を出力する。
-l　：ローカルノードのシステム名を出力する。

uusend　リモートシステムへのファイルの転送 BSD

【書式】 uusend ［*option*］ *sfile sys1!sys2!・・・!sysn!dfile*
【解説】 リモートシステム上にファイルを転送する。
-m *mode*　：リモートシステム上でのファイルモード（*mode*）を設定する。

uustat　uucp の状況出力 SVR

【書式】 uustat ［*options*］
【解説】 uucp のジョブの状況を出力し、ジョブを制御する。
-a　：キューにあるすべてのジョブを対象とする。
-m　：すべてのマシンのアクセス可能状況を出力する。
-k*id*　：ジョブ（*id*）を強制終了させる。
-r*id*　：ジョブ（*id*）の修正時刻を現在時刻にする。

uuto　他の UNIX システムへのファイルの転送 SVR

【書式】 uuto ［*options*］ *source dest*
【解説】 UNIX システム間でファイル（*source*：元、*dest*：先）を転送する。
-p　：いったん、スプールディレクトリにコピーする。
-m　：コピー完了のメールを送信する。

uux　　他の UNIX システムでのコマンドの実行　SVR BSD

【書式】 uux ［*options*］［*sys!*］*cmds*

【解説】 指定したシステム（*sys*）でコマンド（*cmds*）を実行する。

-n　：ユーザに通知しない。

wait　　プロセスの終了待ち　SVR BSD

【書式】 wait

【解説】 すべてのプロセスが終了するのを待って、異常終了に関する出力をする。

xstr　　C言語プログラム中の共有文字列の出力　BSD

【書式】 xstr *file*

【解説】 C言語のソースファイル（*file*）から共有文字列を出力する。

yacc　　コンパイラコンパイラ　SVR BSD

【書式】 yacc ［*options*］ *file*

【解説】 コンパイラコンパイラを実行する。

-d　：y.tab.h を作成する。

-v　：y.output を作成する。

yes　　文字列の繰り返し出力　BSD

【書式】 yes ［*str*］

【解説】 指定した文字列（*str*：省略時は y）を繰り返し出力する。

付録C

UNIX 用語解説

▶記号

.cshrc

Cシェルが起動時に自動的に実行するコマンドファイル。

.login

Cシェルユーザがログインする際に、自動的に実行されるコマンドを記述するファイル。

.logout

Cシェルユーザがログアウトする際に、自動的に実行されるコマンドを記述するファイル。

/dev

特殊ファイルが格納されているディレクトリ。端末をはじめとした各種の入出力装置に対するデバイスファイルがある。

/dev/null

ここに送られたデータは無視される。

/etc

システムの運用管理に使われるファイルが格納されているディレクトリ。

/etc/passwd

ユーザ管理情報を格納したファイル。

/lib

C言語などのプログラミング言語で参照するサブルーチンライブラリのファイルを格納しているディレクトリ。

/sys

カーネルのソースコードが格納されているディレクトリ。

/tmp

テンポラリ（一時）ファイルを格納するために利用するディレクトリ。

/usr

システム関係のツールやドキュメントをはじめとして各種のファイルを格納する、汎用的に用いられるディレクトリ。

/usr/spool

スプーリングされたファイルを格納するために用いられるディレクトリ。

▶B

BSD（*Berkelry Software Distribution*）

1977年以来カリフォルニア大学バークレイ校で独自に開発されているUNIX。最新版は4.3 BSD。viエディタ、Cシェルをはじめとして仮想記憶機能、ジョブ制御、ネットワーク機能、プロセス間通信などはBSDで実現された機能で、現在のAT&T版UNIXのSystemVにはこれらはすべてが取り込まれている。最終版となる4.4 BSDが1993年中には提供される予定。

Bシェル（*Bourne shell*）

UNIXに標準搭載されている最も標準的なシェル。

▶C

core

実行中のプロセスが異常終了したときに、プロセスイメージを格納するファイルの名前。dbx コマンドなどを使ってデバッグする際に利用される。

C言語（*C Language*）

AT&T ベル研究所の T.Dennis，M.Ritchie，Ken Thompson によって開発された UNIX システム記述用の言語。低水準のアセンブリ言語のもつ柔軟性と高級言語のもつ記述性を合わせもつ特徴があるため、現在では汎用言語として広く普及している。

Cシェル（*C shell*）

C言語のシンタックスに似たスクリプト言語仕様をもつシェル。エイリアス、ヒストリ、ジョブ制御などの特徴をもち、Bシェルとともに最もユーザが多い。

▶E

Emacs

MIT の Richard Stallman によって開発されたフルスクリーンエディタ。名称は、editor macros からつけられた。当初は Lisp 言語専用エディタであったが、現在はC言語をはじめとして汎用的に使われる。UNIX には標準エディタとして vi があるが、Emacs はその機能の豊富さや拡張性の高さから、システム開発分野のエンジニアの多くが標準的に使っている。vi との大きな違いは、入力と編集のモードが分かれていない点にある。特徴としては、キーバインディングの設定、マルチウィンドウ・マルチバッファによる複数ファイルの同時編集、カスタマイズ機能などがある。

Ethernet

同軸ケーブルを使った 10 Mbps のバス型 LAN。1975 年に XEROX、DEC、

Intelの3社によって開発され、UNIXワークステーションのLANでは標準的に採用されている。LANの標準規格IEEE 802.3の基になった。

EUC (*Extended Unix Code*)

UNIXで使われる日本語文字コードのうちの1つ。シングルシフト文字を使って、ASCII文字、漢字、カタカナ、外字を共存させたコード体系になっている。

・ASCIIコード ……8ビット目0
・漢字コード ……2バイトとも8ビット目が1

UNIXではEUCコードのほかには、パソコンでも一般的なJISコードやシフトJISコードも使われる。

▶G

GNU (*Gnu is Not Unix*)

Richard Stallmanが設立したFSF (Free Software Foundation) が無償配布しているフリーソフトウェアの総称。Emacs、Cコンパイラ、C++コンパイラ、AWKなど数多くの優秀なソフトがある。

▶I

iノード (*i node*)

UNIXのファイルシステム管理を行うための内部構造体。一つのiノードにはファイルの属性情報すなわちモード、型、所有者名、サイズ、位置関係の情報が書き込まれている。ファイルの内容は保持していない。このiノードを集めた一覧は、UNIXのシステムの周辺に一括して存在している。

▶J

JUNET (*Japanese University/UNIX NETwork*)

日本の代表的な大学や研究機関のUNIXマシンを結合したネットワーク。uucp

をベースとした電子メールや BBS が、研究者間のコミュニケーションに役立っている。WIDE という名称は、TCP/IP プロトコルによるネットワーク部分のことを指している呼び方。

▶M

Mach (*Multiple Asynchronously Communicating Hosts*)

カーネギーメロン大学で開発されている分散オペレーティングシステムのこと。4.3 BSD の互換機能、マイクロカーネルなどの特徴をもつ。OSF/1 や NeXT OS のベースとなっている。

MIT (*Massachusetts Institute of Technology*)

マサチューセッツ工科大学の略称。広い分野で世界をリードする研究が行われている。コンピュータサイエンスもその一つで、人工知能研究をはじめとして、Emacs、Lisp、X Window System などの基本技術が MIT の成果である。

Motif, OSF/Motif

OSF が採用している X Window System ベースのユーザインタフェースの標準仕様。

MULTICS

1964 年に MIT で開発が開始された巨大な TSS システムで、その後の OS の基本技術の基礎を確立したという意義が大きい。1969 年から開発が開始された UNIX にも反面教師になったという意味でも大きな影響を与えた。

▶N

NIS (*Network Information Service*)

従来は YP と呼ばれていた機能。パスワードやホスト名を管理するシステムのファイルの管理を行うための機能を提供している。

NFS（*Network File System*）

サンマイクロシステムズが開発した、LAN上で複数のマシンが同じファイルを共有するためのネットワークプロトコル。同じLAN上であれば、自分の使っているマシン以外のマシンに接続されたディスク上にあるファイルでも、自分のマシンに接続されたディスク上のファイルであるかのように取り扱える。

▶O

OPEN LOOK

UI(UNIX International)が採用しているユーザインタフェースの標準仕様。OPEN LOOK仕様に準拠した代表的なウィンドウシステムとしては、サンマイクロシステムズのOpen Windowsがある。

OSF（*Open Software Foundation*）

UIに対抗する目的で、IBMを中心としてDEC、HP、シーメンス、フィリップス、日立などが参加しているオープンシステム(UNIX)の標準化団体。1988年5月に設立された。ここからOSF/1というOS仕様を共通的に使用する。

▶R

RS-232C

300～19200 bpsという比較的低速なデータ転送を行うために使用されるシリアルインタフェース仕様。ホストコンピュータと端末、モデムなどを接続するさいなどに使われることが多い。

▶S

SCCS（*Source Code Control System*）

プログラム開発においてソースコードのバージョンを自動的に管理してくれるシステム。古いバージョンのソースコードでも任意に復元することができる。

SystemV Release4.0

UI（UNIX International）が採用している UNIX の標準仕様。

▶T

TSS（*Time Sharing System*）

CPU 時間を細かく区切って複数のプログラムに割り当てることによって、あたかもそれらが同時に実行されているかのような状況を作り上げる方式。

▶U

UI（*UNIX International*）

AT&T を中心として、サンマイクロシステムズ、富士通、東芝などが参加している UNIX の標準化団体。OSF と対立関係にある。SystemV という UNIX を共通に使用する。

▶X

X Window System

MIT の Athena プロジェクトの成果としてパブリックドメイン化しているウィンドウシステム。現在では、ワークステーションの標準ウィンドウとしてほとんどのハードウェアメーカーに採用されている。クライアントサーバ方式を採用していて、ネットワークを介して別のマシン上に自分のマシンで実行した結果を表示する、というようなこともできるようになっている。

▶Y

YP（*Yellow Page*）

システム管理のなかでも/etc/passwd や/etc/group などのユーザ管理に使用するファイルの管理を、ネットワーク上で一括管理することによって容易にするための機能。ネットワーク上の1台のマシンにユーザ登録しさえすれば、他

のマシンにはいちいち登録しなくてもすべてのマシンを使うような設定ができる。管理ファイルの一貫性を維持するためにも重要な機能。

▶イ

イーサネット

➡ Ethernet

イエローページ

➡ YP

▶ウ

ウィンドウシステム（*window system*）

OS上でウィンドウ表示を担当するソフトウェアシステム。UNIXでは、X Window Systemが標準的に使用されているが、このほかにもサンマイクロシステムズのNeWS、カーネギーメロン大学のAndrewなどがある。

▶エ

エイリアス（*alias*）

ユーザが自ら指定したコマンド行に付ける別名。Cシェルのaliasコマンドを使って設定する。

▶オ

オブジェクトファイル（*object file*）

コンパイラが出力するマシン語命令で記述された実行形式のプログラムファイル。いくつかのオブジェクトファイルをリンケージエディタで結合した結果として作成されることもある。

オンラインマニュアル（*online manual*）

ハードコピーマニュアルと同じ内容のものが、man コマンドを使うことによって端末やワークステーション上の画面に表示させることができる。man -k でコマンド名が不明でも、関連したキーワードからコマンドを検索することもできる。

▶カ

カーネル（*kernel*）

オペレーティングシステムの核の部分。ハードウェア依存性が高いため、低レベル言語で記述されるが、UNIX の場合にはこの部分を極力C言語で記述しているため移植性が高くなっている。

カレントディレクトリ

➜ワーキングディレクトリ

環境ファイル（*environment files*）

ログインまたはシェルが起動する際に、各種の環境設定を自動的に行うために参照されるファイル。C シェルには、起動時に .login と .cshrc、終了時に .logout があり、B シェルには起動時の .profile がこれに相当し、ユーザの各ホームディレクトリが置かれる。

この他にも、様々なソフトウェアが起動するときに必要な初期設定ファイルがある。ex エディタや vi エディタには .exrc、Emacs エディタには .emacs、X には .Xdefaults というファイルがある。

環境変数（*environment variables*）

個人の作業環境を設定するための変数。setenv コマンドを使って設定するが、起動エディタの種類 (EXINIT)、ターミナルタイプ (TERM) などがある。これらの値は、プロセスが動作するときに参照される。

▶ク

クライアントサーバ（*client-sever*）

単一の処理をその処理を依頼するクライアントと受けるサーバに分散する方式。ネットワークファイルシステム(NFS)やX Window Systemなど、UNIXの代表的なサブシステムはこの方式を採用している。

グループ（*group*）

ユーザが属する部門やプロジェクト、行う作業内容などによって分類し、これを単位としてファイルなどへのアクセス権を設定する。グループファイル /etc/group とパスワードファイル /etc/passwd にユーザとグループの対応関係が記述される。

▶コ

コマンド（*command*）

システムに対して、各種の動作を要求するための命令。UNIXには、入力するコマンド行の書式が標準化されていて、

コマンド名　オプション　引数

となっているため、比較的覚えやすい。

▶シ

シェル（*shell*）

ユーザのコマンド入力とUNIXカーネルを仲介するコマンドインタプリタ。代表的なシェルに、Cシェル、Bシェル、Kシェルなどがある。

シェル変数（*shell variables*）

シェルが動作する際の環境を設置するための変数。setコマンドで設定する。

シンボリックデバッガ（*symbolic debugger*）

ソースプログラムとの対応関係を確認しながらデバッグすることができるツール。代表的なものに dbx コマンドや sdb コマンドがある。

シンボリックリンク（*symbolic link*）

ファイルシステムの枠を越えて別のファイルを参照できるようにしたリンク機能。ln -s コマンドを使って設定できる。

ジョブ（*job*）

一般には計算機で処理を行うひとまとまりの一連の作業単位。C シェルでの取扱いは、1つのコマンド行によって入力されたコマンド群を1ジョブとして扱う。コマンド1個は最低1個のプロセスを生成するので、ジョブは一般に複数のプロセスで構成される。

▶ス

スーパーユーザ（*superuser*）

UNIX のシステム管理を行うための特別なユーザ資格。通常、root というユーザ名でログインするか、一般のユーザ名でログインした後に、su コマンドで資格を得るかのどちらかの方法をとる。スーパーユーザは、

- OS、アプリケーション、I/O 装置のインストールと環境整備
- 日常の運用（起動から終了まで）
- 障害時の対応
- Q/A

などの作業を担当し、全ユーザが快適に利用できるように努める役割を負う。

スタック（*stack*）

いちばん最後に格納されたものから取り出されるデータの格納方式。C シェルにはディレクトリスタック機能があり、pushd コマンドと popd コマンドで制御できるようになっている。

スプーリング（*spooling*）

ジョブの行うプリンタ出力をはじめとした入出力処理を、ジョブの実行と並行して行うこと。UNIXのプリンタ出力では、lprコマンドを使って指定されたファイルのスプーリングを行えば、lpdデーモンがこれを処理して出力するという手順になっている。

スプライン（*spline*）

与えられた点列をもとに補間点を計算し、スムーズな曲線を得るために使われる自由曲線関数。splineコマンドを使えば、計算結果を得ることができる。

▶セ

セッション（*session*）

ログインしてからログアウトするまでの一連の作業。UNIXはマルチタスクOSであるため、一人のユーザが複数のセッションを開設して作業をすることもできる。

絶対パス（*absolute path*）

ルートディレクトリを起点としたパス。

▶ソ

相対パス（*relative path*）

ワーキングディレクトリを起点としたパス。

ソケット（*socket*）

プロセス間通信の方式の一種。ネットワーク上にある異なるマシンで実行されているプロセスとの通信が可能である。ソケット関連のシステムコールを使って実現でき、ファイルシステム上ではls- lコマンドのファイルの種類の位置にsと表示されるものがこれに相当する。

▶テ

テキストエディタ（*text editor*）

テキストファイルを作成するためのユーティリティシステム。UNIX では、標準装備されているものとして、行エディタの ex、スクリーンエディタの vi、ストリームエディタの sed が使えるようになっている。最も仕様頻度の高いスクリーンエディタとして、最近は Emacs エディタの人気が高い。

ディレクトリ（*directory*）

ファイルシステムを階層構造化するために使われるファイルをひとまとまりにするためのファイル。ディレクトリ中には一般ファイルだけでなくディレクトリも置くことができる。ls コマンドを使えば、ディレクトリの内容を確認することができる。

デバイスドライバ（*device driver*）

OS と I/O 装置の間のデータ転送を担当するプログラム。UNIX では、/dev というディレクトリにファイルとして各種 I/O 装置に対応するデバイスドライバが存在している。

デバッガ（*debugger*）

プログラムのエラーすなわちバグを発見するために使われるユーティリティシステム。変数内容の表示、ブレークポイントの設定、トレースなどの機能があり、対象となるプログラムを動作させながら、さまざまな検証を行うことができる。UNIX には、アセンブラレベルのデバッグを行う adb、ソースプログラムとの対応をとりながらデバッグできる BSD 系の dbx や SystemV 系の sdb などがある。

デフォルト（*default*）

明示的な指定がない場合に暗黙的に採用される動作や値のこと。コマンドリファレンス編では、省略時というかたちで表記している。

デーモン（*daemon*）

ネットワークやスプーリングの処理を行うプロセスのように、起動後処理すべきデータが発生するまで待機状態でいて、データの発生に際して即応するプロセス。

電子メール（*electronic mail*）

ネットワークを介した端末やコンピュータ間で電子化された情報をやり取りする手段。UNIXでは、mailコマンドを使って実現できる。

▶ト

特殊ファイル（*special file*）

➜/dev

▶ネ

ネットワークファイルシステム（*network file system*）

➜ NFS

▶ハ

ハードリンク（*hard link*）

➜リンク

バックグラウンド（*background*）

ジョブまたはプロセスが対話的な実行状態にないこと。

パーティション（*partition*）

ファイルシステムを格納するディスク上の区画。UNIXでは、ファイルシステムをその果たす役割に応じていくつかのパーティションに分ける。このパーティションの配置や空き領域の状況を知るには、dfコマンドを使えばよい。

パイプ（*pipe*）

あるコマンドの標準出力と別のコマンドの標準出力を直接結合してデータの転送を行う方式。コマンド行中ではパイプの記号「|」の両わきに2つのコマンドを記述する。

パス（*path*）

各ファイルを特定するためのファイルシステム内における経路。「／」で区切られたディレクトリのリストと最後尾に付けられるファイル名で構成される。ルートディレクトリからの経路を示す絶対パスとワーキングディレクトリからの経路を表す相対パスがある。

パスワード（*password*）

システムを使おうとするユーザが本人であるかどうかを確認するための暗号。ログイン時にユーザ名の入力に続いてパスワードを入力する。

パスワードファイル（*password file*）

各ユーザのパスワードをはじめとした属性情報を格納しているファイル。パスワードは暗号化されているので、画面表示は無意味な文字列であるように見える。/etc/passwd のこと。スーパーユーザにだけ変更が許されている。

パターン（*pattern*）

ファイル中の文字(列)を検索する際の指定に使う形式。grep コマンドや awk コマンドでは正規表現が使われる。

▶ヒ

ヒストリ（*history*）

以前に実行させたコマンド行を記録しておき、これを再利用する機能。Cシェルでは、単に以前のコマンド行を参照するだけでなく、その一部を変更して実行させる機能もある。シェル変数 history に記録する最大行数を設定すれば、そのセッションの間有効である。セッションを跨いでも有効にするためには、

シェル変数savehistに最大行数を設定すればよい。.historyファイルに保存され、次回のセッションでも参照できるようになっている。

標準エラー出力（*standard error output*）

コマンド（プログラム）がエラーメッセージを出力するストリーム。パイプで連結されたコマンドラインの中で、エラーメッセージがなくならないように、標準出力とは別にしている。

標準シェル（*standar shell*）

➡Bシェル

標準出力（*standard output*）

多くのコマンドが出力を行う。

標準入力（*standard input*）

多くのコマンドが入力を受け付ける。

▶フ

ファイル（*file*）

UNIXのファイルには、データの保持のために使われる通常ファイルのほかに、ディレクトリファイル、特殊ファイルなどがある。

フィルタ（*filter*）

標準入力からデータを受け付け、標準出力からデータを書き出す形式のコマンド(プログラム)。パイプで結合したコマンド行では、フィルタが連続的に実行されていくことになる。

フォアグラウンド（*foreground*）

プロセスまたはジョブが対話的な実行状態にあること。端末を占有する優先権をもっている。

プロセス (*process*)

実行状態にあるプログラム。ps コマンドでプロセスの状態を表示することができる。

プロンプト (*prompt*)

システムがユーザに対してコマンドを入力受付状態にあることを示すメッセージ。C シェルでは%、B シェルでは$、スーパーユーザでは#がデフォルトのプロンプトである。変更するには、C シェルでは変数 prompt、B シェルでは変数$PS の設定をすればよい。

▶ヘ

別　名

→エイリアス

▶ホ

ホームディレクトリ (*home directo:y*)

ユーザがログインした時点でワーキングディレクトリとして設定されるディレクトリ。ここを起点として、各ユーザは自分のファイルやディレクトリを配置することができる。どのディレクトリに移動していても、cd コマンドをオプション指定無しで実行すれば、ホームディレクトリに戻ることができる。

ホスト名 (*host name*)

マシンを識別するための固有名。LAN 環境などで、別のマシンとのコミュニケーションをする rlogin コマンド、rcp コマンドなどでは引数に指定する。

▶マ

マウント (*mount*)

パーティションに分割されたファイルシステムを結合して、使用可能状態に

すること。通常、スーパーユーザがmountコマンドを使って作業を行う。

マジック番号（*magic number*）

実行ファイルが、もともとはコンパイラの出力なのかシェルスクリプトなのかを判別するために使われる番号。

マルチタスク（*multitask*）

2つ以上のプログラム（プロセス）を同時に実行すること。

マルチユーザ（*multiuser*）

同時に二人以上のユーザが使えるシステム。

▶メ

メール（*mail*）

UNIXでは、mailコマンドとその関連コマンドを使って電子メールの交換をすることができる。

➜電子メール

▶ユ

ユーザ名/ユーザID（*user name/user ID*）

ユーザ名はユーザを識別するために使われる文字列で、ログイン時に入力する時点で使用する。ログイン名とも呼ぶ。ユーザIDは、UNIXが内部的にユーザの識別に使用している番号。idコマンドの出力で、両方とも確認することができる。

▶リ

リダイレクション（*redirection*）

標準入力、標準出力、標準エラー出力をデフォルトの端末から別のデバイス

に変更すること。

リンク（*link*）

ディレクトリに格納されているファイル名とｉノードの結合関係。ln コマンドを使えば新たなリンクを作成でき、リンクを削除するときには rm コマンドを使えばよい。

リンク数（*link count*）

ファイルがいくつのディレクトリからリンクされているかを表す数。通常、ディレクトリファイルでは、自分自身からのリンクと親のディレクトリからのリンクの最低リンク数は 2 となる。ls -l コマンドの表示で確認することができる。

▶ル

ルートディレクトリ（*root directory*）

ファイルシステムの頂点にある最上位のディレクトリ。

▶ロ

ローカルエリアネットワーク

→ LAN

ログアウト（*logout*）

UNIX へのアクセスを終了するときに行う手続き。logout コマンドか CTRL ＋ D の入力を行う。これによって、他人が無断で使用することができなくなる。

ログイン（*login*）

UNIX へのアクセスを開始するときに行う手続き。login : プロンプトに続けて、ユーザ名とパスワードを正確に入力する。

ログインシェル（*login shell*）

ログインしたときに最初に起動されるシェル。finger コマンドの出力で確認することができる。chsh コマンド、または passwd -s コマンドで任意に変更することができる。

▶ワ

ワーキングディレクトリ（*working dorectory*）

コマンドの実行時にその時点で対象となるディレクトリ。カレントディレクトリともいう。cd コマンドを使って適宜変更することができるが、通常ログイン直後はホームディレクトリがワーキングリレクトリとなっている。

ワイルドカード（*wild card*）

シェルが解釈する特殊な意味をもった文字。? や * が代表的なワイルドカードで任意の文字列として解釈する。

付録D

MINIX コマンド対応表

MINIX オプションに説明のないものは、UNIX コマンドと同じ機能をもつ。

UNIX コマンド	MINIX コマンド	MINIX コマンドで使えるオプション
at	at	なし
awk	bawk awkのサブセット (basic awk)	なし
banner	banner	なし
basename	basename	なし
cal	cal	なし
cat	cat	-u
cc	cc	-c,-o,-w,-S そのほか多数あり
chgrp	chgrp	なし
chmod	chmod	なし
chown	chown	なし
clear	clr	なし
cmp	cmp	-l,-s
comm	comm	-1,-2,-3
cp	cp	なし
cut	cut	-c*list*,-f*list*,-d*chr* -i：連続するデリミタを１つとして処理する

UNIX コマンド	MINIX コマンド	MINIX コマンドで使えるオプション
date	date	-q：標準入力から日付を読み出す
dd	dd	なし
df	df	なし
diff	diff	なし
du	du	-a, -s -l *n*：指定したレベル(*n*)まで出力する
echo	echo	-n
ed	ed	なし
(emacs)	elle Emacs ライクなエディタ	なし
ex	ex	-R：読み出し専用 -e：ex エディタをエミュレートする -t：指定されたタグで編集する -v：vi エディタをエミュレートする
fgrep	fgrep	-c, -f *file*, -h, -l, -n, -s, -v
file	file	なし
find	find	なし
fold	fold	-n
grep	prep	-e *expression*, -l, -n, -s, -v
head	head	-n
id	id	なし
indent	indent	一般的なオプションはほぼ可
kill	kill	-signal
last	last	-n, -f *file* -r：前回のリブートに遡って出力
leave	leave	なし
ln	ln	なし

UNIXコマンド	MINIXコマンド	MINIXコマンドで使えるオプション
login	login	なし
lpr	lpr	なし
ls	ls	-A, -C, -F, -R, -a, -d, -f, -g, -i, -l, -r, -s, -t, -u
mail	mail	-f *file*, -p, -q, -r, -v -d：シェル変数MAILERの使用を強制する
make	make	-f, -i, -k, -n, -p, -q, -r, -s, -t
man	man	なし
mkdir	mkdir	なし
more	more	-d, -f, -l, -s, -u -p：画面をスクロールさせない
mv	mv	なし
nroff	nroff	-m -b：バックスペース可能な装置 -v：nroffのバージョンを印刷する -*n*：印刷する最初のページ（*n*） +*n*：印刷する最後のページ（*n*）
od	od	-b, -c, -d, -o, -x -h：16進数表示
passwd	passwd	なし
paste	paste	-d*list*, -s
pr	pr	-f, -h *string*, -l n, -n, -t, -w *n* -M：MINIX方式の行番号を使う
printenv	printenv	なし
ps	ps	-a, -l, -x -U：システム実行形式からデータベースを更新する
pwd	pwd	なし

UNIX コマンド	MINIX コマンド	MINIX コマンドで使えるオプション
rm	rm	-f, -i, -r
rmdir	rmdir	なし
sed	sed	-e *script*, -f *file*, -n -g：すべての置換コマンドにグローバルフラグを付ける
sed	gres sed ライクなストリームエディタ	-g：各行ごとに最初の対象だけを置換する
sh	sh version 7 の B シェルのほとんどの機能を備えるシェル	なし
size	size	なし
sleep	sleep	なし
sort	sort	-c, -d, -f, -i, -m, -n, -o *file*, -r, -t c -u：同じ行があれば削除する
split	split	-n
stty	stty	なし
su	su	なし
tail	tail	-c, -l, -*num*
tar	tar	c, o, f, t, x v：冗長モード F：エラー後も強制的に続行
tee	tee	-a, -i
tr	tr	-c, -d, -s
tty	tty	-a
uniq	uniq	-c, -d, -u, -*n*, +*n*
users	users	なし

UNIX コマンド	MINIX コマンド	MINIX コマンドで使えるオプション
vi	elvis 　vi から派生したエディタ	-R：読み出し専用 -e：ex エディタをエミュレートする -t：指定されたタグで編集する -v：vi エディタをエミュレートする
wc	wc	-c, -l, -d
whereis	whereis	なし
which	which	なし
who	who	なし
whoami	whoami	なし
write	write	-c：cbreak モードを使用する -v：メッセージを出力しながら動作させる

付録E

CTRL キーの操作

CTRL + ¥……コマンドの実行を中断し、プログラムイメージをcoreファイルに書き出す。

CTRL + c……実行されているコマンドの中止

CTRL + d……入力の終了またはログアウト

CTRL + o……実行しているコマンドからの出力だけを停止

CTRL + q……画面表示の再開（CTRL + s……の後の再開）

CTRL + s……画面表示の一時停止

CTRL + u……コマンド行全体の削除

CTRL + w……コマンド行の最後の語の削除

CTRL + z……プロセスの実行の一時停止

索　引

コマンド索引

用語索引

●記号

●A

●B

●C

●か行

●さ行

●た行

●な行

●は行

●ま行

●や行

●ら行

●わ行

著者略歴

舟本　奨（ふなもと　すすむ）

早稲田大学卒業後、富士通株式会社を経て、現在は(学)電子学園教員。特種情報処理技術者。
『人工知能概論』（専門教育出版：共著)、『Prolog詳説』（啓学出版：共訳)、『MS-Windows3.0基本操作ガイド』（ナツメ社：監修)、『パワーユーザのためのUNIXコマンド活用ハンドブック』（パーソナルメディア：監修)、『UNIX基本操作ガイド(上)コマンド編』（技術評論社）など多数の著書がある。

実用UNIXハンドブック

著　者　舟本　奨

発行者　田村正隆

発行所　株式会社　ナツメ社
東京都千代田区神田神保町1-52加州ビル2F（〒101）
電話　03(3291)1257（代表）
振替　00130-1-58661

制　作　ナツメ出版企画株式会社
東京都千代田区神田神保町1-52加州ビル3F（〒101）
電話　03(3233)8961

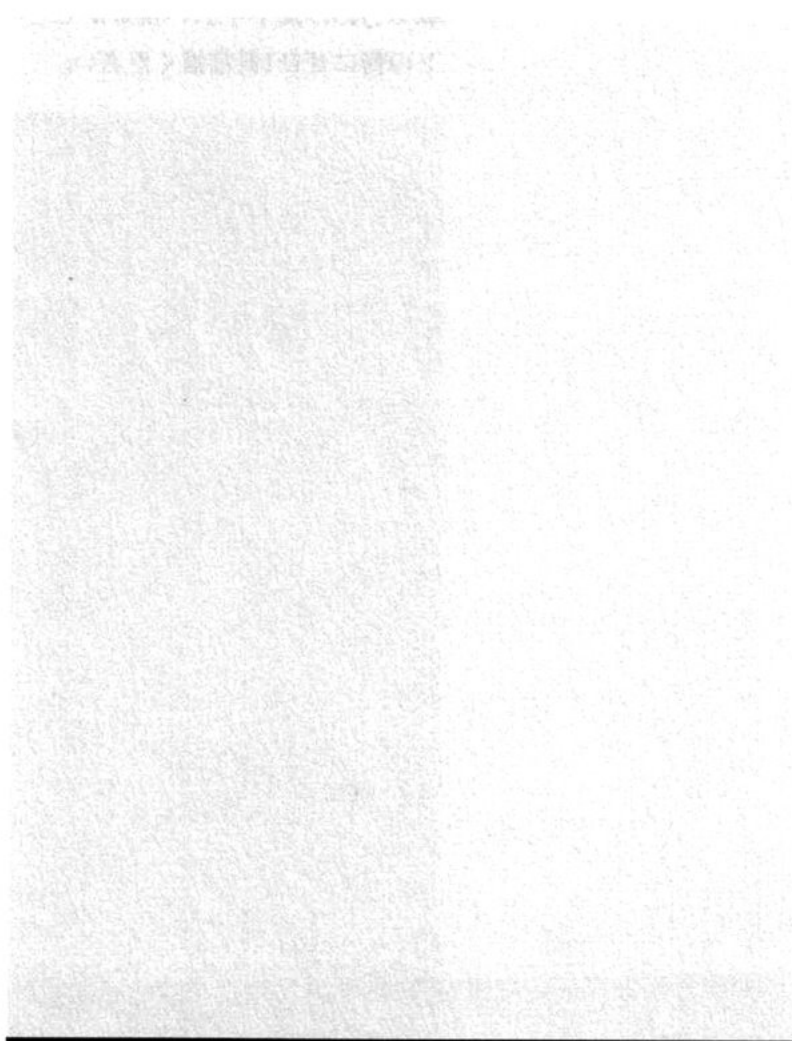
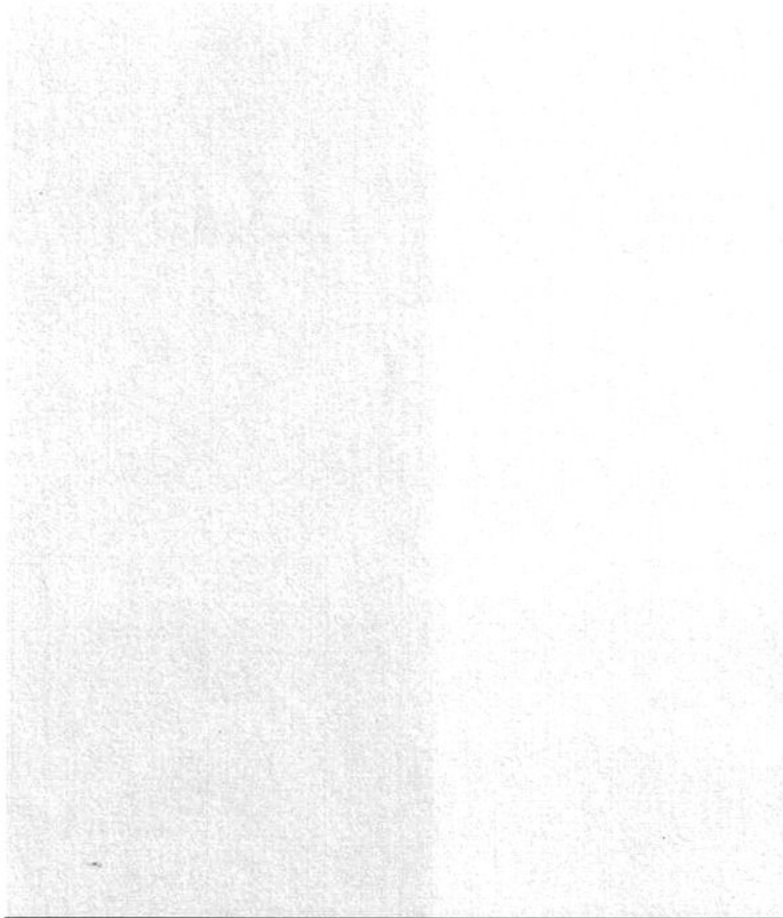
印　刷　ラン印刷社

ISBN4-8163-1654-X　Printed in Japan

わかりやすいコンピュータ用語辞典

高橋三雄監修——定価1300円

ISBN4-8163-0890-3

本書は、汎用コンピュータのみにとどまらず、情報処理、通信、ニューメディアまで、広範囲の用語を網羅した辞典です。写真と図版を豊富に使用し、コンピュータに関してはこれ一冊ですべてがわかるようにとの編集方針にもとづいて、わかりやすく解説しました。すべての用語を50音順に配列してありますので、英語、日本語に関係なく、一度で目的の用語を引くことができます。机上に一冊常備し、何か疑問が生じたときにお役立てください。

パソコン・ワープロ漢字辞典

上柿力編――定価1860円

ISBN4-8163-0696-X

本書は、JISの新版(1983年)に準拠し、第1・第2水準の漢字を3種類の索引から検索できるように編集してあります。また、機種等によっては第2水準漢字を扱ったり印字したりできないものもありますので、この区別がすぐわかるよう2色刷にしました。また、巻末に漢字・非漢字一覧、主要姓名用漢字および新旧のJISで字体が異なった主要漢字、コンピュータを使ってのJISコードから区点およびシフトJIS、ASCIIコードへ変換をするプログラムなどを掲載し、JISの漢字を広い視野からとらえました。ワープロ、パソコンの傍にぜひ1冊常備ください。

実用UNIXハンドブック
[改訂新版]

発行———1995年8月20日
著者———舟本 奨
発行者——田村正隆
発行所——株式会社ナツメ社
東京都千代田区神田神保町1-52 加州ビル2F
郵便番号101
電話(03)3291-1257
振替 00130-1-58661
制作———ナツメ出版企画株式会社
東京都千代田区神田神保町1-52 加州ビル3F
郵便番号101
電話(03)3233-8961
定価———2300円
〈落丁・乱丁本はお取り替えします〉

ISBN4-8163-1654-X

C2055 P2300E

ナツメ社|定価2,300円

(本体2,233円)

H a n d b o o k

■本書は好評を得た「実用UNIXハンドブック」の改訂新版です。
今回の改訂でのポイントはつぎのようになっています。

●基本操作解説の増補
●コマンド・リファレンスへのコマンド追加
●対応UNIX製品の追加(3製品から10製品)
●エディタ・リファレンスへのEmacsの追加

また、つぎのような内容も新たに加わりました。

●第2レベルコマンド・リファレンス
●UNIX用語解説
●MINIXコマンドとの対応表

もちろん、そのほかにも

●Cシェルコマンド
●vi操作法
●機能から引ける充実した索引

などは従来どおり掲載。大幅な改訂をおこなうことにより、
これまで以上に、初心者ユーザーから中級・上級ユーザー
それぞれの必要性に応えられる内容となっています。
UNIXユーザー必携の実用ハンドブック、
堂々の再登場です。